AUTOMATED VIGILANCE SYSTEM FOR CLASH DETECTION IN LOCALIZED ENVIRONMENTS USING MOBILENETV3

A PROJECT REPORT

Submitted by

SIVAGAMA SUNDARI.V

SOWMIYASREE.S

SWATI.S

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2023

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"AUTOMATED VIGILANCE SYSTEM FOR CLASH DETECTION IN LOCALIZED ENVIRONMENTS USING MOBILENETV3"** is the bonafide work of "**SIVAGAMA SUNDARI.V (211419104255), SOWMIYASREE.S (211419104263), SWATI.S (211419104283)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.L.JABASHEELA, M.E., Ph.D.,	**Mrs.D.JENNIFER, M.E., (Ph.D.),**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
	ASSOCIATE PROFESSOR
DEPARTMENT OF CSE,	DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,	PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,	NASARATHPETTAI,
POONAMALLEE,	POONAMALLEE,
CHENNAI-600 123.	CHENNAI-600 123.

Certified that the above candidates were examined in the End Semester Project Viva-Voce Examination held on 11/04/2023

INTERNAL EXAMINER **EXTERNAL EXAMINER**

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **“AUTOMATED VIGILANCE SYSTEM FOR CLASH DETECTION IN LOCALIZED ENVIRONMENTS USING MOBILENETV3”** is the bonafide work of “**SIVAGAMA SUNDARI.V (211419104255), SOWMIYASREE.S (211419104263), SWATI.S (211419104283)”** who carried out the project work under my supervision.

SIGNATURE
Dr.L.JABASHEELA, M.E., Ph.D.,
HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE
Mrs.D.JENNIFER, M.E., (Ph.D.),
SUPERVISOR
ASSISTANT PROFESSOR

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidates were examined in the End Semester Project Viva-Voce Examination held on 11/04/2023

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.,** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHI KUMAR, M.E., Ph.D.** and **Dr.SARANYASREE SAKTHI KUMAR B.E., M.B.A., Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.,** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr.L.JABASHEELA , M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank my Project Guide **Mrs.D.JENNIFER, M.E., (Ph.D.)** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

SIVAGAMA SUNDARI.V

SOWMIYASREE.S

SWATI.S

ABSTRACT

The idea to automate vigilance system for physical clash detection using MobileNetV3, a light weight neural network architecture and computer vision techniques, for continuous monitoring in real time helps to provide safety and security to people. The main objective to design this system is to detect potential physical clashes among people in the surrounding and automate a voice call to the authority to notify that a physical clash event is going on. The system is trained on MobileNetV3 large model for which real time violence situations video dataset is used. The model's detection of physical clashes is tested through various video inputs, for which it achieved 96% accuracy. In real time, the saved weights of the model is used through live camera feed, and a voice call alert is automated using Twilio API to the registered authority, by which significant steps at the earliest can be carried out to avoid further physical clashes. The system can be deployed using surveillance cameras. Thus, the proposed system serves as a significant step in the field of automated surveillance and can be considered as a valuable contribution for providing enhanced security. Overall, this paper demonstrates the potential of automated surveillance system and an efficient way in monitoring places for physical clash detection.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

ABBREVIATION	EXPANSION
CNN	Convolutional Neural Network
LOC	Lines Of Code
KLOC	Thousand Lines Of Code
ER	Entity-Relationship
RLVS	Real Life Violence Situations
UML	Unified Modeling Language
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
GTD	Global Terrorism Database
NIBRS	National Incident-Based Reporting System
CV	Computer Vision

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Violence is a term hearing which makes us feel uncomfortable, there are different kinds of violence which have different levels of impact on the society and the people involved. It is the use of physical force or aggression to cause harm, injury, or damage to oneself, others, or property and can take many forms, including domestic violence, child abuse, sexual assault, hate crimes, and terrorism. Physical clash is a complex issue that can be influenced by individual, social, cultural, and systemic factors. It can have significant physical, emotional, and psychological effects on victims, families, communities, and society as a whole. Efforts to prevent and reduce violence often involve a multi-faceted approach that includes education, intervention, enforcement, and support for victims.

One of the ways to prevent it is by video surveillance, emergency response by automated alert system. Video surveillance is a widely used technology to monitor public spaces, homes, and workplaces. Video cameras can capture visual information and provide an objective record of events. With the increasing number of video cameras in public spaces, video analysis techniques are needed to extract useful information from the large amounts of video data generated. Violence detection in video surveillance is an important application of video analysis that can help to prevent violent incidents and promote public safety. The goal of violence detection in video surveillance is to automatically identify and classify violent events in real-time. This requires the development of algorithms and models that can analyse video data and identify violent behaviour accurately and efficiently. Different approaches have been proposed for violence detection, including machine learning, deep learning, and computer vision techniques.

1.2 PROBLEM DEFINITION

The problem definition for clash detection using MobileNetV3 can be stated as follows:

With the increasing number of violent incidents worldwide, there is a growing need for automated systems that can quickly detect and prevent violent acts. Currently, many surveillance systems rely on human monitoring, which can be time-consuming, costly, and prone to errors. Therefore, there is a need for an automated system that can accurately detect violence and alert the authorities in real-time.

The automated vigilance system for violence detection should be able to analyse live video feeds from cameras installed in public areas, schools, and workplaces, and detect any violent incidents such as physical assault, gunshots, or even verbal threats. The cause of clashes includes clashes within families, offenders, criminals, killers, fights between locals, harassments, gang activities in local environments. The major consideration to safety should be given to home alone people, old-age people, ladies hostels, domestic places, public places, war zone.

The system should be able to accurately distinguish between violent and non-violent events and send alerts to the relevant authorities in real-time. Additionally, the system should be able to store data for future analysis and tracking of suspicious individuals or activities.

Overall, the aim of this project is to develop an efficient and reliable automated vigilance system for violence detection that can help prevent violent incidents and improve public safety.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

J. C. Vieira et al, [4] discusses a method for detecting violence using a low-cost convolutional neural network (CNN) on an embedded system. The authors argue that traditional methods for detecting violence are often computationally expensive and not suitable for low-cost embedded systems. The proposed method utilizes a smaller CNN model, which is trained and tested on a dataset of videos containing violence and non-violence using transfer learning. The paper presents the results of experiments conducted to test the effectiveness of the proposed method on different types of violence, variations in lighting conditions, and different camera angles. The authors report high accuracy rates for detecting violence in these scenarios.

V. D. Huszár et al, [13] presents a method for detecting violent events in real-time using machine learning algorithms. The authors argue that traditional methods for detecting violence in video surveillance applications are often computationally expensive and not suitable for real-time use. The proposed method utilizes a feature extraction algorithm to extract relevant features from the video stream and a machine learning algorithm to classify the events as violent or non-violent. They also present the results of experiments conducted to test the effectiveness of the proposed method on a dataset of videos containing violent and non-violent events. The authors report high accuracy rates for detecting violent events in these scenarios.

Y. Watanabe et al, [15] provides a method for detecting violence in videos using salient features. The authors argue that traditional methods for detecting anomalies are often computationally expensive and not suitable for real-world applications. The proposed method utilizes a deep neural network to extract salient features from videos and a classification algorithm to classify the events as normal or anomalous. They also present the results of experiments conducted to test the effectiveness of the proposed method on a dataset of videos containing normal and anomalous events. The authors report high accuracy rates for detecting anomalies in these scenarios and argue that the proposed method is effective for real-world anomaly detection in video surveillance and other applications.

K. B. Kwan-Loo et al, [7] proposes a method for detecting violent behaviour using neural networks and pose estimation. The authors argue that traditional methods for detecting violence in surveillance videos are often limited in their ability to accurately identify violent behaviour. The proposed method utilizes pose estimation to extract relevant features from video frames, and a convolutional neural network to classify the behaviour as violent or non-violent. They also present the results of experiments conducted to test the effectiveness of the proposed method on a dataset of videos containing violent and non-violent behaviour. The authors report high accuracy rates for detecting violent behaviour in these scenarios and argue that the proposed method is effective for real-time detection of violent behaviour in video surveillance applications.

M. G. Constantin et al, [8] presents a benchmark dataset for evaluating violent scene detection algorithms in multimedia. The authors argue that current methods for detecting violent scenes are often limited in their ability to accurately identify such scenes due to the lack of a standardized dataset for evaluating the performance of different algorithms. The proposed benchmark dataset consists of a large collection of videos annotated with labels indicating the presence or absence of violent scenes. The authors also describe the process of designing and optimizing a convolutional neural network for detecting violent scenes in the dataset. The authors present the results of experiments conducted to test the effectiveness of the proposed method on the benchmark dataset. They report high accuracy rates for detecting violent scenes.

CHAPTER 3

SYSTEM

ANALYSIS

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Violence detection using MobileNetV2 is a deep learning approach that uses a MobileNetV2 neural network architecture to detect violent content in images or videos. MobileNetV2 is a lightweight neural network architecture specifically designed for mobile devices and embedded systems, which makes it well-suited for applications where computational resources are limited. To use MobileNetV2 for violence detection, the neural network is first trained on a dataset of labelled images or videos that contain violent content and non-violent content.

The network learns to identify patterns and features in the input data that distinguish between the two classes. During the testing phase, the trained network takes an input image or video frame and computes a probability score indicating the likelihood that the input contains violent content. If the probability score is above a certain threshold, the input is classified as containing violence.

Disadvantages of the existing system:

One disadvantage of using MobileNetV2 for violence detection is that it may not perform as accurately as larger, more complex neural network architectures. Disadvantage of using MobileNetV2 for violence detection is that it may not generalize well to different types of violent content or to different cultural contexts. For example, the types of violent behaviour that are considered acceptable or unacceptable may vary across different cultures, and the dataset used to train the neural network may not reflect this diversity. In summary, while MobileNetV2 is a lightweight and efficient neural network architecture that is well-suited for mobile and embedded devices, its capacity and generalizability may be limited when used for violence detection.

3.2 PROPOSED SYSTEM

Violence detection using MobileNetV3 is another deep learning approach that uses a MobileNetV3 neural network architecture to detect violent content in images or videos. MobileNetV3 is an upgraded version of MobileNetV2, which incorporates several improvements and optimizations to achieve better accuracy and efficiency. The primary advantage of using MobileNetV3 for violence detection over MobileNetV2 is its improved performance. MobileNetV3 uses a number of advanced techniques, such as efficient channel attention, hard-swish activation function, and mobile inverted bottleneck blocks, to achieve better accuracy and efficiency compared to MobileNetV2.

The approach is similar to that of MobileNetV2. The neural network is first trained on a dataset of labelled images or videos that contain violent content and non-violent content. The network learns to identify patterns and features in the input data that distinguish between the two classes. During the testing phase, the trained network takes an input image or video frame and computes a probability score indicating the likelihood that the input contains violent content. If the probability score is above a certain threshold, the input is classified as containing violence. Compared to MobileNetV2, the advantages of using MobileNetV3 for violence detection include better accuracy, faster computation, and higher efficiency. MobileNetV3 is also designed to be more flexible and adaptable, with multiple variants optimized for different tasks and constraints. In summary, violence detection using MobileNetV3 is a state-of-the-art deep learning approach that offers several advantages over MobileNetV2, including improved performance and efficiency.

Open Computer Vision was used to capture live feed, then it is converted to frames which were detected using this model. If violence is detected, an automated voice call is generated to the registered authority using Twilio API and the frame is saved locally to the system.

3.3 FEASIBILITY STUDY

The objective of feasibility study is not only to solve the problem but also to acquire a sense of its scope. During the study, the problem definition was crystallized and aspects of the problem to be included in the system are determined. Consequently, benefits are estimated with greater accuracy at this stage. The key considerations are:

Economic feasibility

Technical feasibility

Social feasibility

Economic Feasibility

Economic feasibility studies not only the cost of hardware, software is included but also the benefits in the form of reduced costs are considered here. This project, if installed will certainly be beneficial since there will be reduction in manual work and increase in the speed of work.

Total number of lines of code (LOC) = 714

KLOC = 714/1000 = 0.714

Effort = 2.4 * (0.714) ^1.05 = 1.684 person-months

Development time = 2.5*(1.684) ^0.38 = 3.047 months

Average staff size = 1.684/3.047 = 0.552 persons

Productivity = 0.714/1.684 = 0.424 KLOC/person-months

P = 424 LOC/person-months

Technical Feasibility

Technical feasibility evaluates the hardware requirements, software technology, available personnel etc., as per the requirements it provides sufficient memory to hold and process.

1) Deep Learning algorithm – Mobile Net V3

2) Artificial Intelligence – Computer Vision

3) Google Drive

4) IDE: Google Collaboratory

Social Feasibility

Social feasibility is a detailed study on how one interacts with others within a system or an organization. Social impact analysis is an exercise aimed at identifying and analysing such impacts in order to understand the scale and reach of the project's social impacts. Social impact analysis greatly reduces the overall risks of the project, as it helps to reduce resistance, strengthens general support, and allows for a more comprehensive understanding of the costs and benefits of the project. The social application of the project is that it ensures the safety of the people in the following environments:

1) Home alone

2) Gang activities

3) Criminal acts

4) War zone

5) Hostels

6) Public places

3.4 HARDWARE ENVIRONMENT

Processor: Intel Core i5

RAM: 512 MB and above

Hard Disk: 40 GB and above

3.5 SOFTWARE ENVIRONMENT

Backend: PYTHON

Frontend: TKINTER

Technology: Deep Learning

Operating System: Windows 7

Tools: Anaconda Navigator /TensorFlow/Spyder/Google Collaboratory

CHAPTER 4

SYSTEM

DESIGN

CHAPTER 4

SYSTEM DESIGN

4.1. ER DIAGRAM

The following figure shows the Entity-Relationship diagram of the project

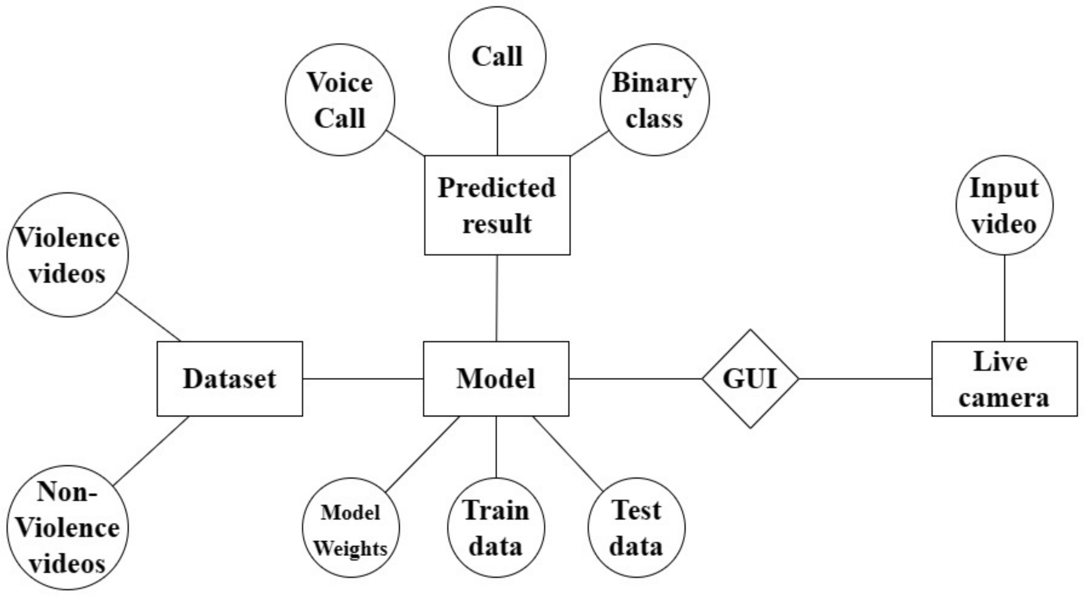


Fig 4.1. ER Diagram

4.2 DATA DICTIONARY

RLVS "Real Life Violence Situations", displays a group of video clips divided into video clips that collected from YouTube videos shows violence and non-violence in different situations and places Dataset Contains 2000 videos of full length 3 hours decided into 1000 videos of (Violence action) contained videos of bare hands fights, non-projectile weapon abuse fights that are sticks, knives, and big throw-able objects.

Generally, the fights are near-distance fights. and 1000 videos of (non-violence action) Contained videos of shaking hands with friends, talking in a café, riding a horse, eating, taking a walk in the park, etc. The purpose of the dataset is to provide a large and diverse set of videos for researchers and engineers working on violence recognition from videos using deep learning techniques. The dataset could also potentially be used for other purposes such as action recognition or scene classification.

The importance of using RLVS dataset in this project are:

1) High resolution dataset with robustness

2) Can be used greatly in fine-tuned algorithms, which enhances the capability of reducing the decision time in violence and crime detection.

4.3 DATA FLOW DIAGRAM

The Data Flow Diagram of the system is shown in **Fig 4.2.**

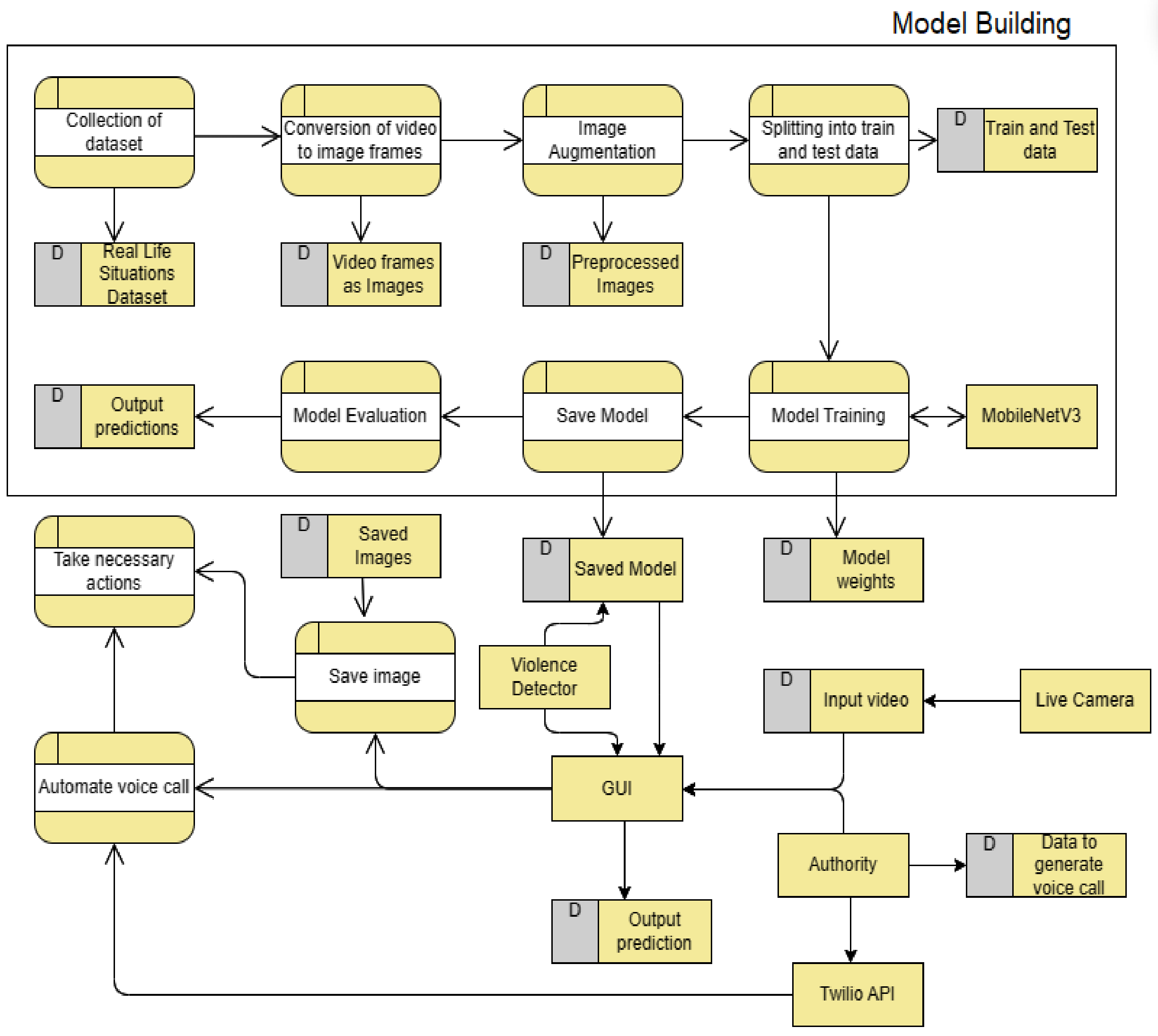


Fig 4.2 Data Flow Diagram of the system

4.4 UML DIAGRAMS

Unified Modeling Language (UML) is a modeling language used in software engineering to visually represent systems, processes, and architectures. UML diagrams are graphical representations that use standardized symbols and notations to communicate complex systems and concepts. The UML diagrams of the system are as follows.

USE CASE DIAGRAM:

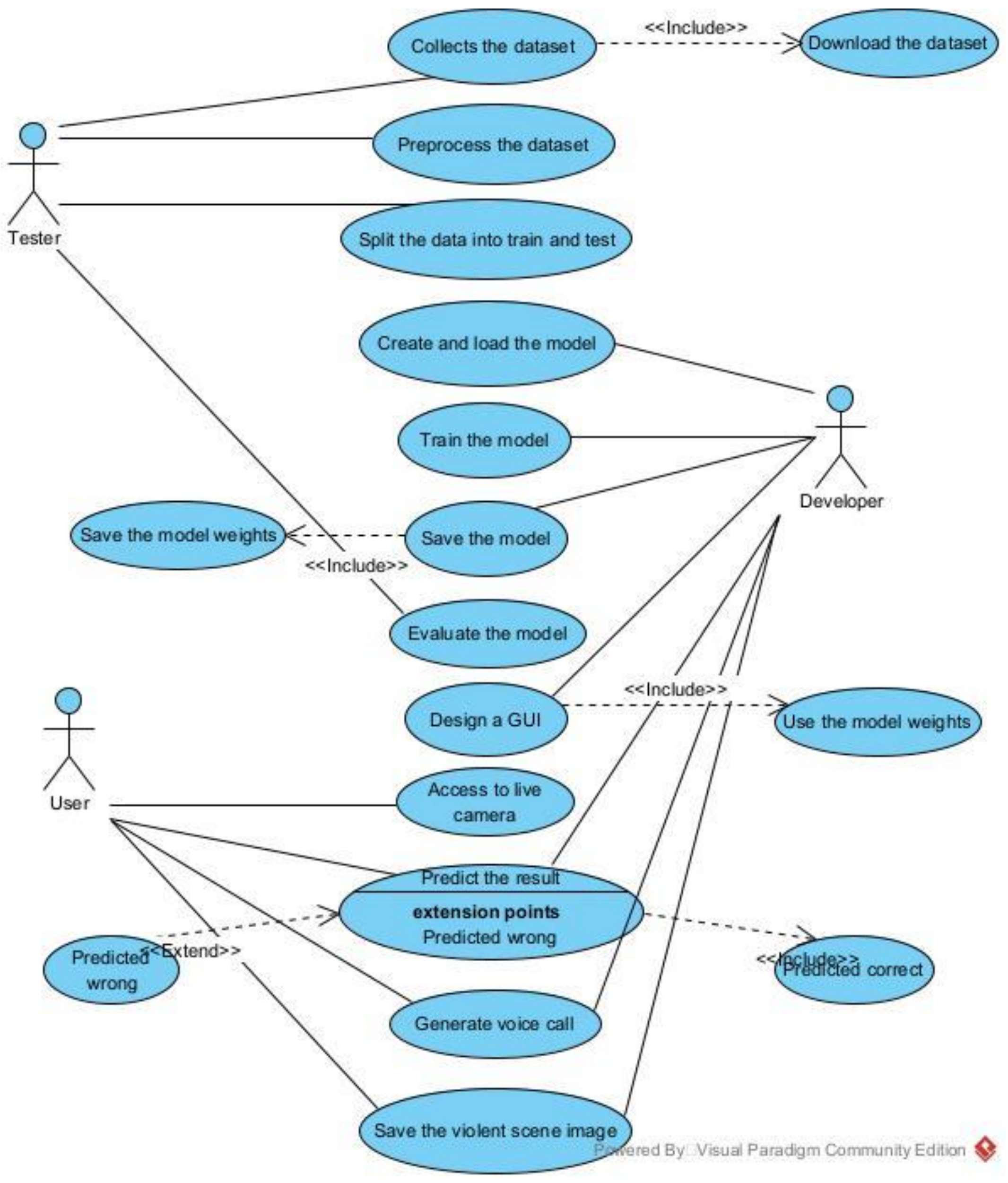


Fig 4.3. Use Case Diagram

CLASS DIAGRAM:

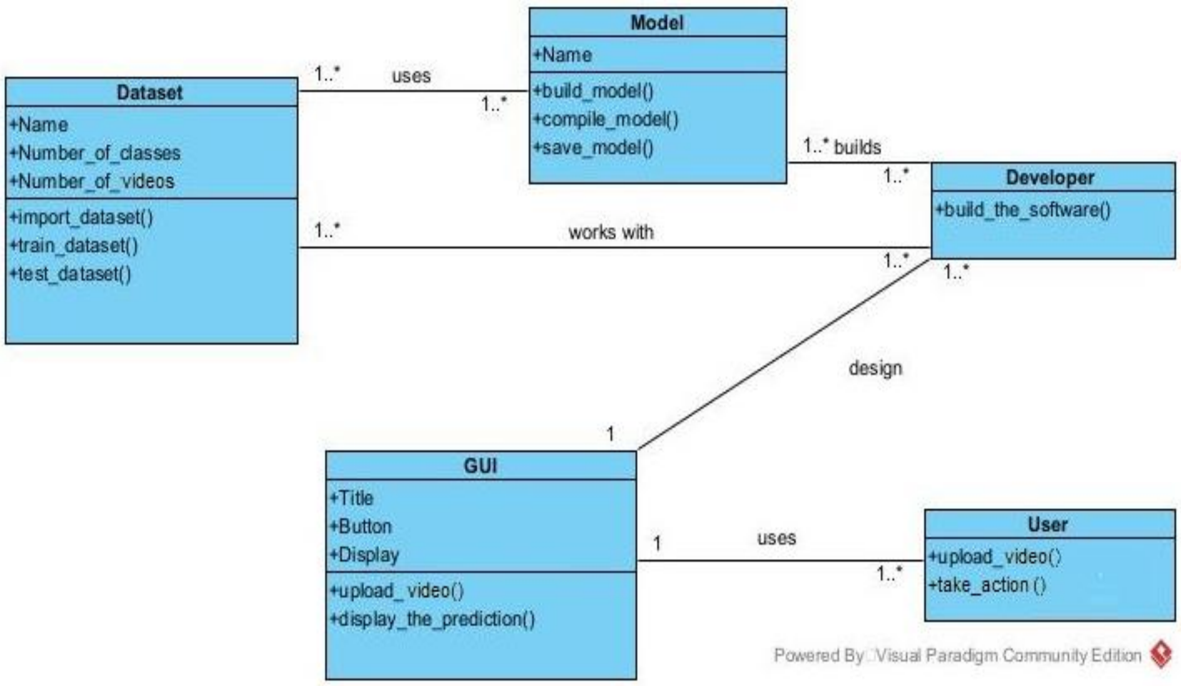


Fig 4.4. Class Diagram

ACTIVITY DIAGRAM:

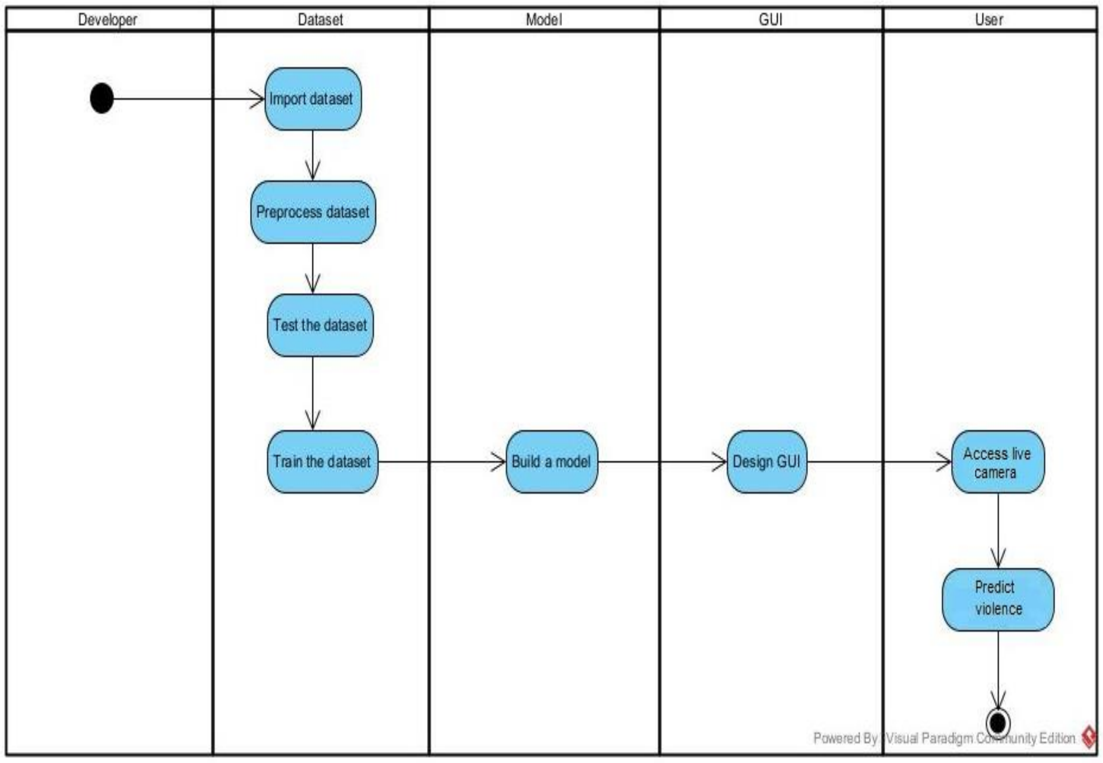


Fig 4.5. Activity Diagram

SEQUENCE DIAGRAM:

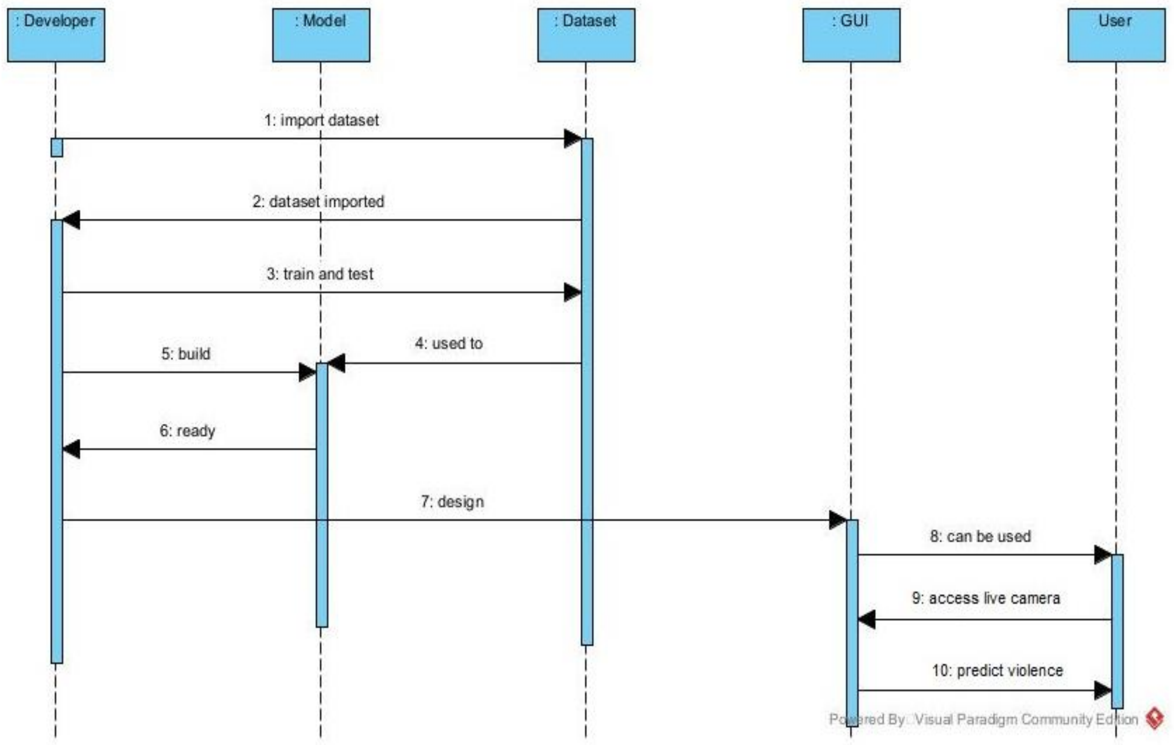


Fig 4.6. Sequence Diagram

COLLABORATION DIAGRAM:

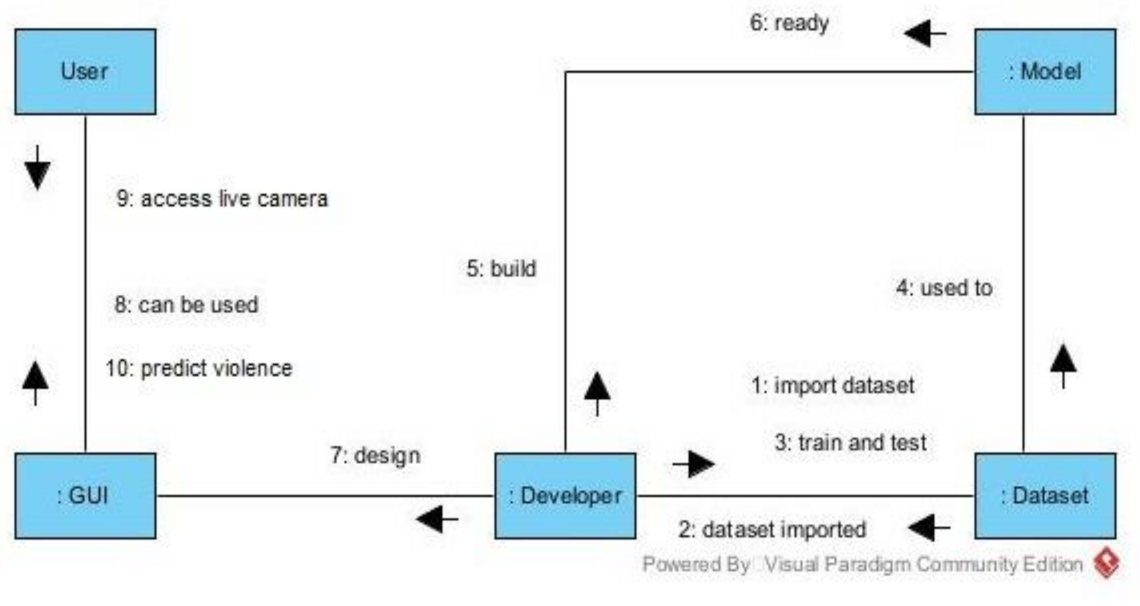


Fig 4.8. Collaboration Diagram

CHAPTER 5

SYSTEM ARCHITECTURE

CHAPTER 5

SYSTEM ARCHITECTURE

ARCHITECTURE OVERVIEW

This displays the whole design of our working model including the components and the states occurring during the execution of the detection process. Violence detection using MobileNetV3 is a neural network architecture that efficiently classifies input images into binary categories of violence or non-violence, with a focus on reducing computational cost and optimising performance for on-device deployment. Key features of MobileNetV3 include depth-wise separable convolutions and linear bottleneck layers to reduce the computational cost of the feature extraction module. The network consists of an input module, a feature extraction module, a classification module, and a post-processing module. The input module pre-processes the input image, while the feature extraction module extracts high-level features. The classification module classifies the features into 2 classes, and the post-processing module generates the output.

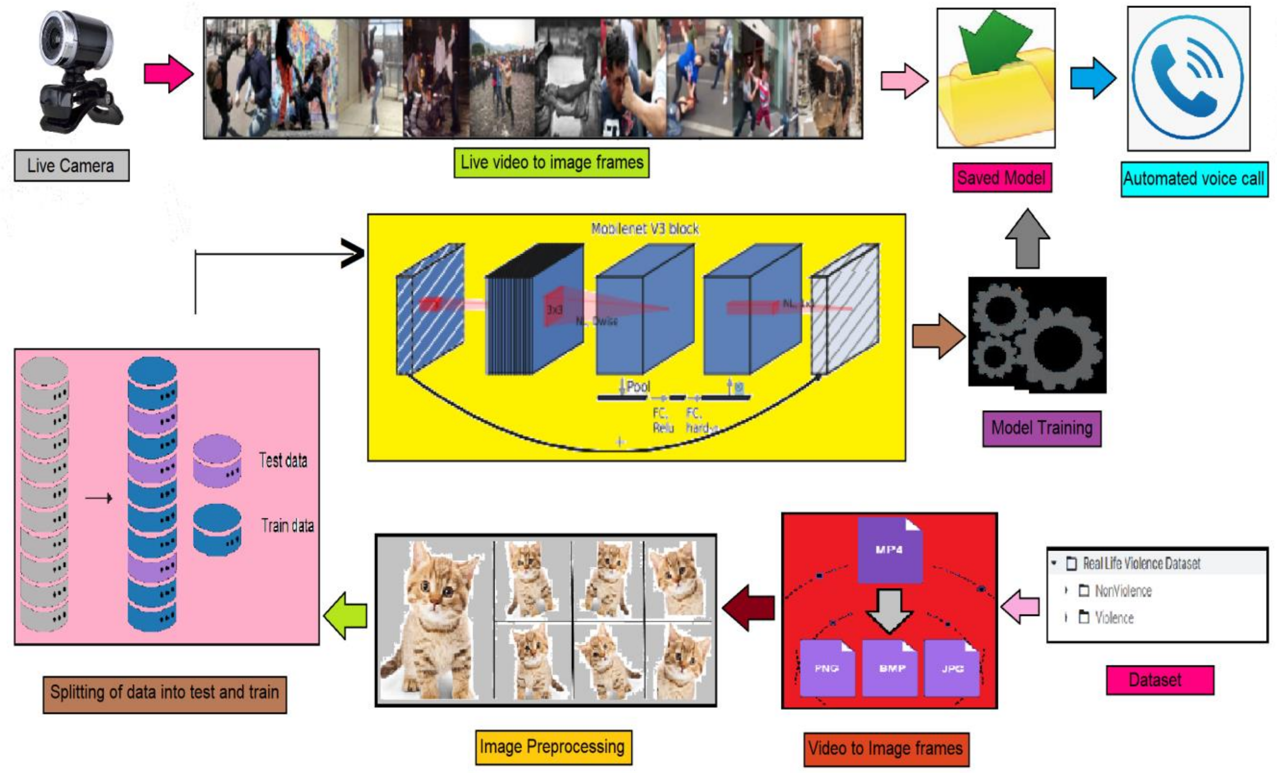


Fig 5.1 Architecture Diagram

5.1 MODULE DESIGN SPECIFICATION

The modules of this project are:

Input Module: The input module of MobileNetV3 is responsible for preparing the input image for processing by the network. It typically includes operations such as resizing, normalization, and data augmentation. Resizing is used to ensure that the image is of a fixed size, while normalization is used to ensure that the pixel values are scaled to a specific range. Data augmentation is used to increase the size of the training dataset by applying random transformations such as rotation, flipping, and cropping. These operations are essential for ensuring that the network is able to handle a wide range of input images.

Feature Extraction Module: The feature extraction module of MobileNetV3 is responsible for extracting high-level features from the pre-processed input image. It consists of multiple convolutional layers that apply a series of filters to the input image to extract relevant features. The architecture of MobileNetV3 is designed to reduce the computational cost of this module by using a combination of depth-wise separable convolutions and linear bottleneck layers. Depth-wise separable convolutions separate the spatial and channel-wise convolutions, reducing the number of parameters and computations required. Linear bottleneck layers further reduce the computational cost by reducing the number of channels while maintaining the quality of the feature maps.

Classification Module: The classification module of MobileNetV3 is responsible for classifying the features extracted by the previous module into different categories. In the case of violence detection, the output of this module will be a binary classification of whether the input image contains violence or not. This module typically consists of fully connected layers that take the feature maps from the previous module and apply a series of linear transformations to produce a probability distribution over the different categories. The final layer usually consists of a soft-max activation function to ensure that the probabilities sum to one.

Post-processing Module: The post-processing module of MobileNetV3 is responsible for performing post-processing on the output of the classification module. In the case of violence detection, this module may perform tasks such as thresholding the output to make a final decision on whether violence is present in the image. The threshold is typically set based on the requirements of the application and the trade-off between false positives and false negatives. For example, if the application requires a high level of accuracy, the threshold may be set high to reduce the number of false positives, even if it results in more false negatives. Conversely, if the application requires a high recall, the threshold may be set low to reduce the number of false negatives, even if it results in more false positives.

Pre-Requisites

This project requires good knowledge of Deep learning, Python, working on Google Collaboratory notebooks, Keras library, Numpy. Make sure you have installed all the following necessary libraries:

- pip install tensorflow
- keras
- twilio
- opencv
- numpy
- tqdm
- Google Collaboratory

Project File Structure

Downloaded from dataset:

- **Violence Dataset –** Dataset folder which contains 1000 violence videos.
- **Non - Violence –** Dataset folder which contains 1000 non - violence videos. The below files will be created by us while making the project.
- **Modelnew.h5 –** It will contain trained models for violence detection.
- **Model.h5 –** This file contains trained models for the alert system.
- **LiveDetection.py –** This python file contains the implementation of the alert system for users.
- **Mobilenetv3_training.ipynb –** Collab notebook in which we train and build our violence detection model.
- **Violence_Testing.ipynb –** Collab notebook in which we train the model to test the video for violence detection.
- **Non-Violence_Testing.ipynb –** Collab notebook in which we train the model to test the video for non-violence detection.

Building The Python Based Project

Let’s start by initializing the Google Collaboratory notebook server by typing Google Collaboratory in the console of your project folder. It will open up the interactive Python notebook where you can run your code. Create a Python3 notebook and name it as Mobilenetv3_training.ipynb. Also google provides the run time in open access where certain features like GPU and TPU runtime are provided for all. Similarly, other files are also created using the above mentioned steps.

Starting with the Collection of Dataset

A dataset typically consists of a set of input features or variables and a corresponding set of output labels or responses. The goal of using a dataset in machine learning is to train a model to make accurate predictions or classifications on new, unseen data. Violence can cause physical injuries, emotional trauma, social isolation, financial strain, and damage to interpersonal relationships, all of which can have lasting effects on individuals and communities. It is important to prevent violence and provide support for those affected by it.

There are some datasets that contain information on violent incidents reported in the news or through other sources.

Global Terrorism Database (GTD)

National Incident-Based Reporting System (NIBRS)

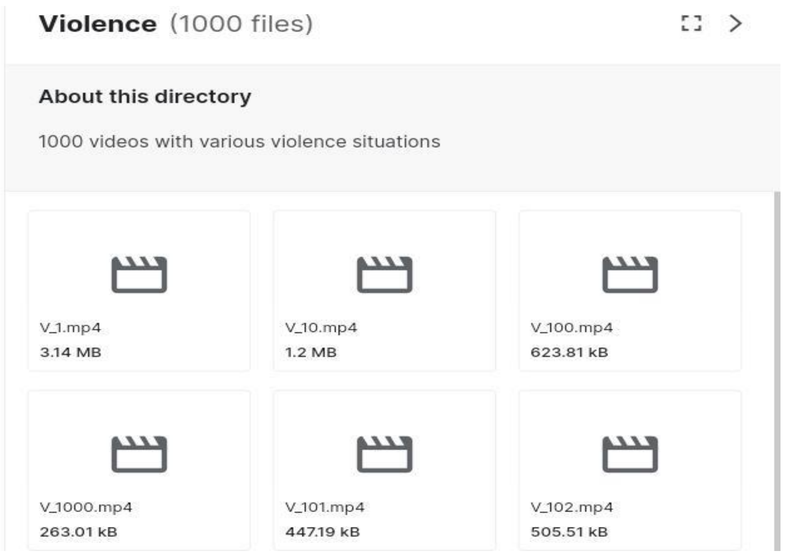
Gun Violence Archive.

Real Life Violence Situation (RLVS)

Real Life Violence Situation (RLVS)

The dataset contains a total of 2,000 videos, with 1,000 videos depicting violence and 1,000 videos depicting non-violent actions. The violence videos were collected from various sources on YouTube and feature real street fight situations in different environments and conditions. The non-violence videos were also collected from YouTube and feature different human actions such as sports, eating, and walking.

It's important to note that the quality of a dataset depends on many factors, including the quality of the data labelling, the diversity of the videos, and the representativeness of the dataset to the target problem. Ultimately, the choice of dataset would depend on the specific needs and goals of the researcher or engineer working on violence recognition. According to the requirements of the project RLVS seems to be well suited as the dataset because it is specifically created for violence recognition using deep learning techniques and these videos are collected from real-world scenarios, which could make the dataset more representative of the types of violence that occur in everyday life.



Violence (1000 files)

About this directory

1000 videos with various violence situations

V_1.mp4
3.14 MB

V_10.mp4
1.2 MB

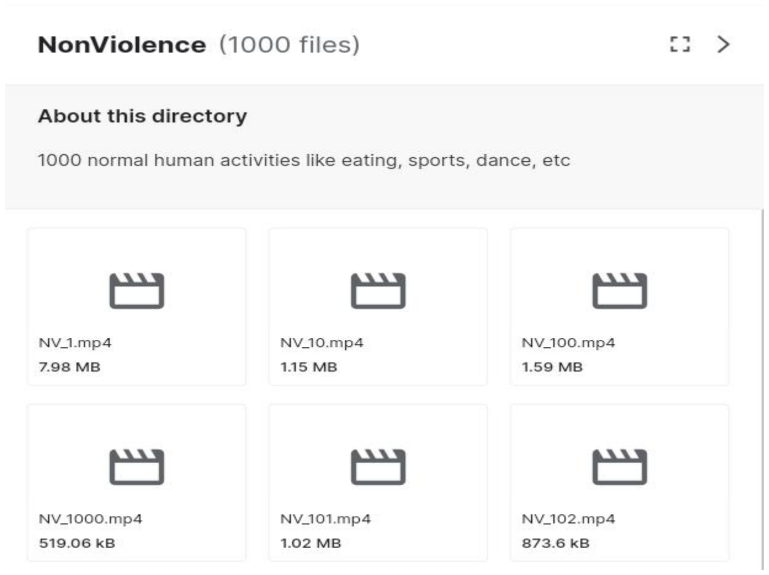
V_100.mp4
623.81 kB

V_1000.mp4
263.01 kB

V_101.mp4
447.19 kB

V_102.mp4
505.51 kB

Fig 5.2. Violence directory files



NonViolence (1000 files)

About this directory

1000 normal human activities like eating, sports, dance, etc

NV_1.mp4
7.98 MB

NV_10.mp4
1.15 MB

NV_100.mp4
1.59 MB

NV_1000.mp4
519.06 kB

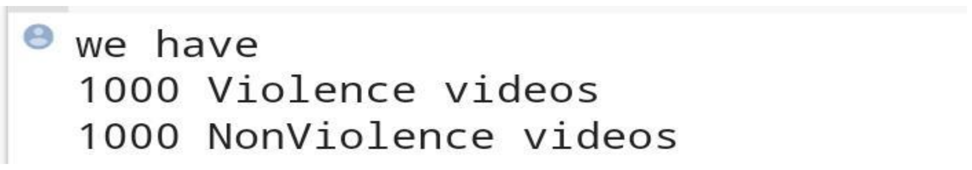
NV_101.mp4
1.02 MB

NV_102.mp4
873.6 kB

Fig 5.3. Non-Violence directory files

Pre-processing

Pre-processing is a critical step in violence detection using MobileNetV3, as it prepares the input images for processing by the network. The goal of pre-processing is to standardize the input images, make them easier to analyze, and help the network achieve better performance. The pre-processing steps typically include resizing, normalization, and data augmentation.

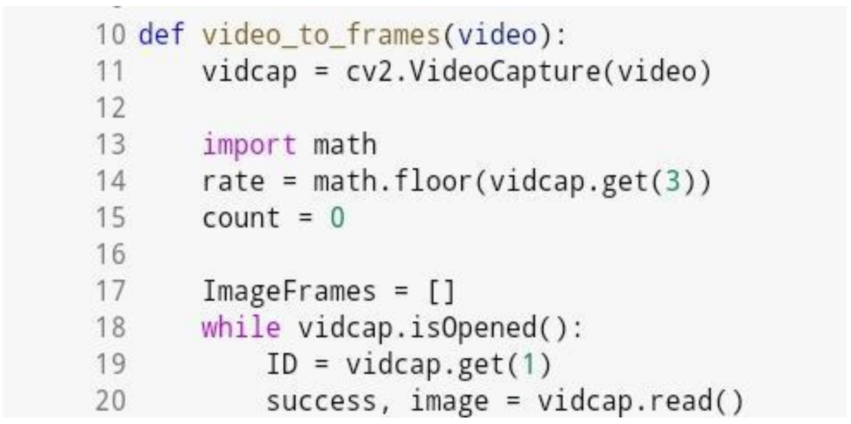


```
we have
1000 Violence videos
1000 NonViolence videos
```

Fig 5.4. Dataset

Converting video to image frames

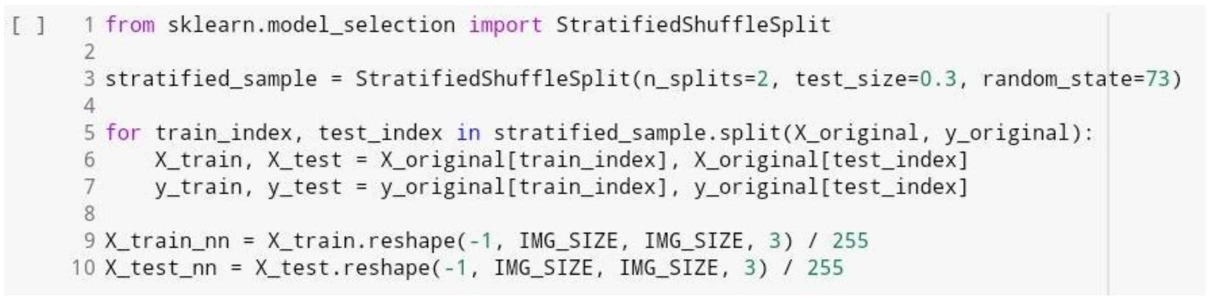
The process of converting videos into image frames involves extracting each frame of the video and saving it as a separate image file. This can be done using a video processing library, such as OpenCV or FFmpeg. The first step is to read in the video file using the library, and then to extract the total number of frames in the video. Once the number of frames has been determined, the video can be iterated over, and each frame can be extracted and saved as a separate image file. Once the frames have been extracted and saved as image files.



```
def video_to_frames(video):
    vidcap = cv2.VideoCapture(video)

    import math
    rate = math.floor(vidcap.get(3))
    count = 0

    ImageFrames = []
    while vidcap.isOpened():
        ID = vidcap.get(1)
        success, image = vidcap.read()
```

Fig 5.5. Code for video to image frame conversion

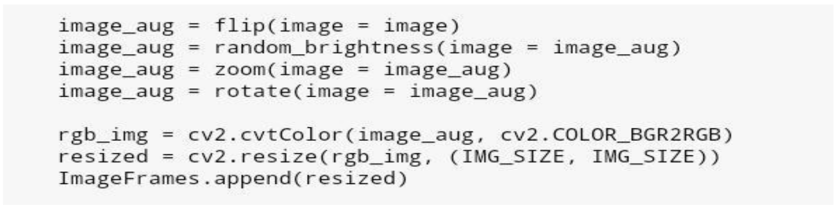
The dataset for violence detection using MobileNetV3 is split into a training set and a testing set to evaluate the network's performance on unseen data. The dataset is randomly shuffled and split based on a percentage split, such as 80% for training and 20% for testing. The training set is used to train the network, while the testing set is used to evaluate its performance. The split percentage can vary depending on the dataset size and problem requirements. Techniques for splitting include manual selection or using libraries like scikit-learn. Splitting the dataset helps avoid overfitting and assess network performance on unseen data.



```
[ ]
from sklearn.model_selection import StratifiedShuffleSplit

stratified_sample = StratifiedShuffleSplit(n_splits=2, test_size=0.3, random_state=73)

for train_index, test_index in stratified_sample.split(X_original, y_original):
    X_train, X_test = X_original[train_index], X_original[test_index]
    y_train, y_test = y_original[train_index], y_original[test_index]

X_train_nn = X_train.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255
X_test_nn = X_test.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255
```

Fig 5.6. Code to split the dataset

Resizing: Resizing is the process of changing the size of the input image. MobileNetV3, like many other neural networks, requires a fixed input size. The input images are usually resized to a fixed size, such as 128x128, to ensure that the network is able to handle images of different sizes. Resizing can be done using various interpolation techniques such as bilinear, bicubic, or nearest neighbour interpolation. The choice of interpolation technique can affect the quality of the resized image, which in turn can impact the performance of the network.



```
image_aug = flip(image = image)
image_aug = random_brightness(image = image_aug)
image_aug = zoom(image = image_aug)
image_aug = rotate(image = image_aug)

rgb_img = cv2.cvtColor(image_aug, cv2.COLOR_BGR2RGB)
resized = cv2.resize(rgb_img, (IMG_SIZE, IMG_SIZE))
ImageFrames.append(resized)
```

Fig 5.7. Code for image augmentation

Normalization: Normalization is used to ensure that the pixel values of the input images are in a specific range. This is typically done by subtracting the mean value of the pixel values and dividing by the standard deviation. Normalization helps to ensure that the input values are not too large or too small, which can affect the performance of the network. In MobileNetV3, the pixel values of the input image are typically normalized to be between -1 and 1.

Data Augmentation: Data augmentation is used to increase the size of the training dataset by applying random transformations to the input images. This helps to reduce overfitting and improve the generalization performance of the network. Common data augmentation techniques include random cropping, flipping, and rotation. In addition, specialized data augmentation techniques may be used for violence detection, such as adding noise or blurring the images to simulate motion.

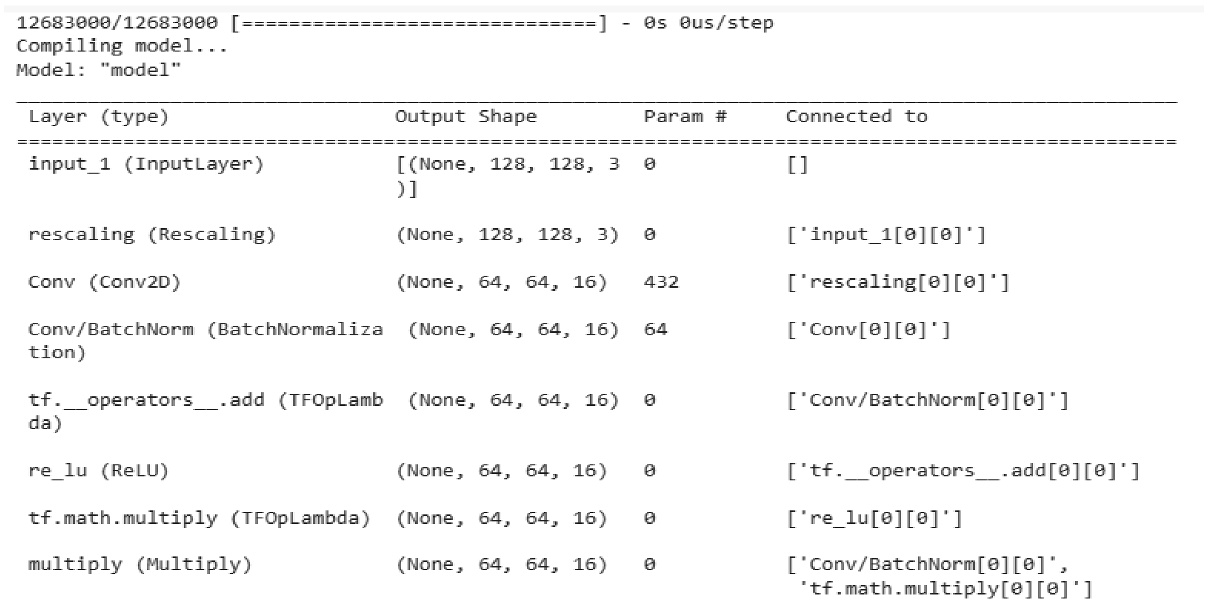
In addition to these pre-processing steps, it may be necessary to pre-process the images to enhance certain features. For example, in violence detection, it may be useful to pre-process the images to enhance the contrast, brightness, or colour saturation, as these features may be relevant in distinguishing violent images from non-violent ones. Overall, pre-processing is an important step in violence detection using MobileNetV3, as it helps to ensure that the network is able to handle a wide range of input images and achieves optimal performance.

Implementation of MobileNetV3: MobileNetV3 is a lightweight and efficient deep learning model designed for mobile and embedded devices. The network architecture is based on a combination of depth-wise separable convolutions, linear bottlenecks, and a feature pyramid network.

The depth-wise separable convolutions reduce the computational cost of the network by separating the convolutional layers into a depth-wise convolution (which applies a single filter per input channel) and a pointwise convolution (which applies a 1x1 convolution to combine the output of the depth-wise convolution).

The linear bottlenecks are used to reduce the dimensionality of the network, which helps to improve the model's accuracy and reduce its memory footprint. The feature pyramid network is used to extract features at multiple scales, which helps the network to capture both local and global information in the input image.

In addition to these architectural features, MobileNetV3 also uses a variety of optimization techniques, such as hard-swish activations, squeeze-and-excitation blocks, and weighted-entropy loss, to further improve the efficiency and accuracy of the model.



```
12683000/12683000 [==============================] - 0s 0us/step
Compiling model...
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 128, 128, 3  0           []
                                )]

 rescaling (Rescaling)          (None, 128, 128, 3)  0           ['input_1[0][0]']

 Conv (Conv2D)                  (None, 64, 64, 16)   432         ['rescaling[0][0]']

 Conv/BatchNorm (BatchNormaliza  (None, 64, 64, 16)  64          ['Conv[0][0]']
 tion)

 tf.__operators__.add (TFOpLamb  (None, 64, 64, 16)  0           ['Conv/BatchNorm[0][0]']
 da)

 re_lu (ReLU)                   (None, 64, 64, 16)   0           ['tf.__operators__.add[0][0]']

 tf.math.multiply (TFOpLambda)  (None, 64, 64, 16)   0           ['re_lu[0][0]']

 multiply (Multiply)            (None, 64, 64, 16)   0           ['Conv/BatchNorm[0][0]',
                                                                  'tf.math.multiply[0][0]']
```

Fig 5.8. Compiling the model

Model Training: The training loop is a critical part of training a deep learning model, including the violence detection model using MobileNetV3. The loop involves several steps, including:

Forward pass: In the forward pass, the input image is passed through the model, and the output of the model is computed.

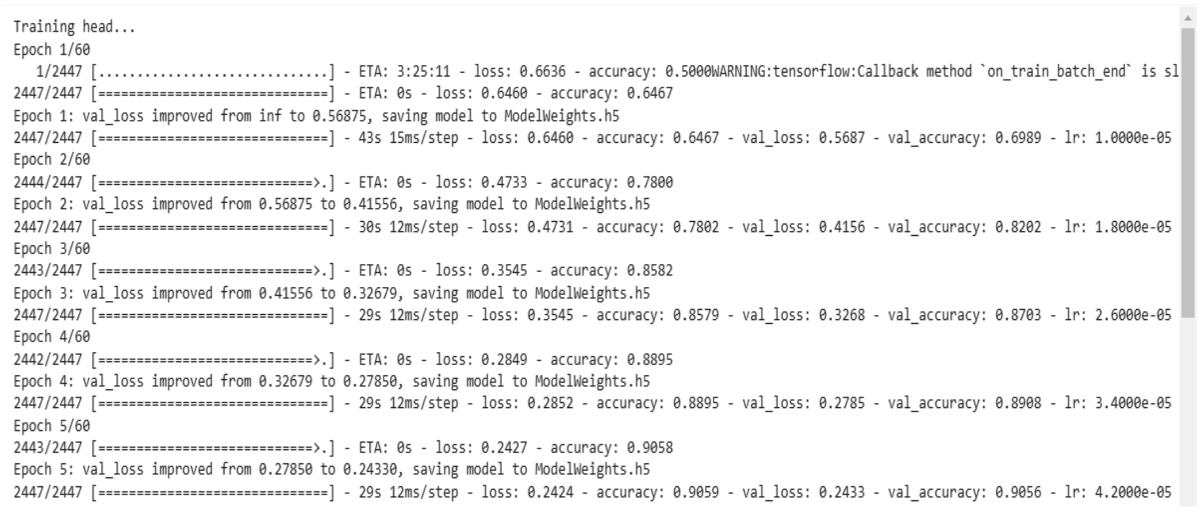
Loss calculation: The output of the model is compared to the ground truth labels to calculate the loss, which represents the difference between the predicted and actual outputs.

Backward pass: The backward pass involves computing the gradients of the loss with respect to the model parameters, which allows the model to update its weights to improve its performance.

Optimization: The model parameters are updated using an optimization algorithm, such as stochastic gradient descent, which adjusts the weights in the direction that reduces the loss.

Repeat: The forward pass, loss calculation, backward pass, and optimization steps are repeated for each image in the training set until the model converges to a set of weights that minimize the loss.

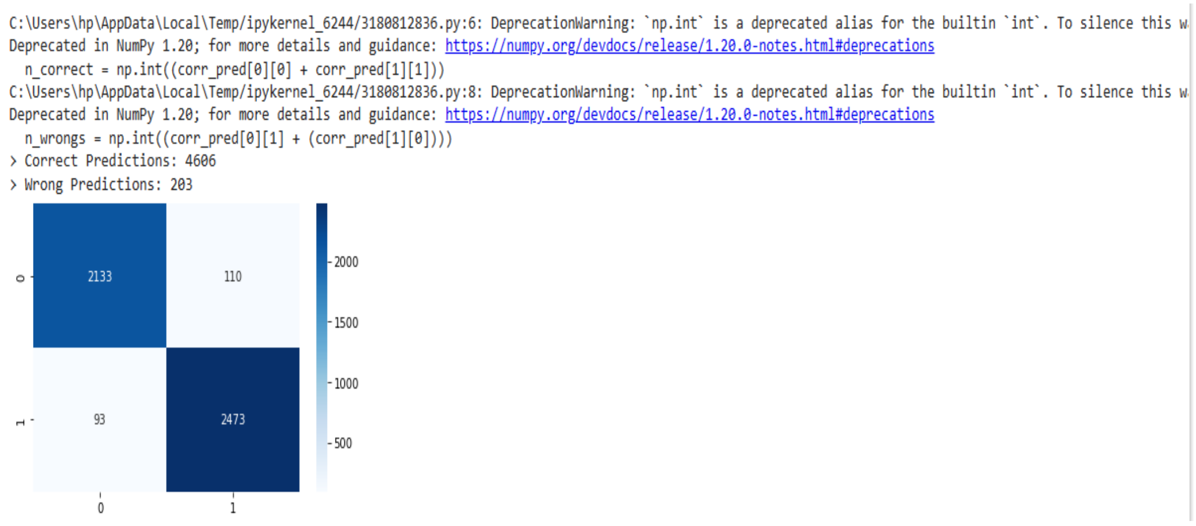
Hyperparameters are parameters that are set before training the model and cannot be learned directly from the data. Examples of hyperparameters include the learning rate, batch size, optimizer, and regularization strength. Tuning these hyperparameters is essential for achieving the best performance of the model.



```
Training head...
Epoch 1/60
   1/2447 [..............................] - ETA: 3:25:11 - loss: 0.6636 - accuracy: 0.5000WARNING:tensorflow:Callback method `on_train_batch_end` is sl
2447/2447 [==============================] - ETA: 0s - loss: 0.6460 - accuracy: 0.6467
Epoch 1: val_loss improved from inf to 0.56875, saving model to ModelWeights.h5
2447/2447 [==============================] - 43s 15ms/step - loss: 0.6460 - accuracy: 0.6467 - val_loss: 0.5687 - val_accuracy: 0.6989 - lr: 1.0000e-05
Epoch 2/60
2444/2447 [============================>.] - ETA: 0s - loss: 0.4733 - accuracy: 0.7800
Epoch 2: val_loss improved from 0.56875 to 0.41556, saving model to ModelWeights.h5
2447/2447 [==============================] - 30s 12ms/step - loss: 0.4731 - accuracy: 0.7802 - val_loss: 0.4156 - val_accuracy: 0.8202 - lr: 1.8000e-05
Epoch 3/60
2443/2447 [============================>.] - ETA: 0s - loss: 0.3545 - accuracy: 0.8582
Epoch 3: val_loss improved from 0.41556 to 0.32679, saving model to ModelWeights.h5
2447/2447 [==============================] - 29s 12ms/step - loss: 0.3545 - accuracy: 0.8579 - val_loss: 0.3268 - val_accuracy: 0.8703 - lr: 2.6000e-05
Epoch 4/60
2442/2447 [============================>.] - ETA: 0s - loss: 0.2849 - accuracy: 0.8895
Epoch 4: val_loss improved from 0.32679 to 0.27850, saving model to ModelWeights.h5
2447/2447 [==============================] - 29s 12ms/step - loss: 0.2852 - accuracy: 0.8895 - val_loss: 0.2785 - val_accuracy: 0.8908 - lr: 3.4000e-05
Epoch 5/60
2443/2447 [============================>.] - ETA: 0s - loss: 0.2427 - accuracy: 0.9058
Epoch 5: val_loss improved from 0.27850 to 0.24330, saving model to ModelWeights.h5
2447/2447 [==============================] - 29s 12ms/step - loss: 0.2424 - accuracy: 0.9059 - val_loss: 0.2433 - val_accuracy: 0.9056 - lr: 4.2000e-05
```

Fig 5.9. Model training

Model Evaluation: Model evaluation is critical for determining the performance of a machine learning model on unseen data. It is important for generalization, model selection, hyperparameter tuning, debugging, and confidence in the model's predictions. Evaluation allows comparison of different models and selection of the optimal model for a task. It also helps to identify issues such as overfitting and underfitting. Evaluation is typically done on a validation set, and the final evaluation is done on a separate test set. Evaluation metrics such as accuracy, precision, recall, and F1 score are commonly used to assess model performance. Model evaluation is an iterative process that may require several rounds of experimentation to achieve optimal results. In summary, model evaluation is essential for building effective machine learning models that can be used in real-world applications.



```
C:\Users\hp\AppData\Local\Temp/ipykernel_6244/3180812836.py:6: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this w
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  n_correct = np.int((corr_pred[0][0] + corr_pred[1][1]))
C:\Users\hp\AppData\Local\Temp/ipykernel_6244/3180812836.py:8: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this w
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  n_wrongs = np.int((corr_pred[0][1] + (corr_pred[1][0])))
> Correct Predictions: 4606
> Wrong Predictions: 203
```

Fig 5.10. Model Evaluation

5.2 ALGORITHMS

CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks (CNNs) are a popular type of neural network used for image classification, object detection, and other computer vision tasks. The architecture of a CNN is inspired by the structure of the human visual cortex, with the aim of enabling computers to interpret images in a similar way to humans. The key building block of a CNN is the convolutional layer. The convolutional layer applies a filter to the input image, producing a feature map. The filter is a matrix of weights that is slid over the input image, computing a dot product at each position. The output of this operation is a single number in the feature map. The filter's dimensions and the stride determine the size of the output feature map. The stride is the number of pixels the filter is moved between computations. A larger stride results in a smaller feature map. Multiple filters can be used to produce multiple feature maps, each capturing different aspects of the input image.

The pooling layer is usually placed after the convolutional layer. It reduces the spatial dimensions of the feature map by taking the maximum or average value of each region of the map. The purpose of this layer is to reduce the number of parameters in the model and prevent overfitting. The pooling layer can be either max pooling or average pooling. Max pooling selects the maximum value in each region, while average pooling calculates the average of the values. The pooling layer also introduces translation invariance, which means that the model can detect the same feature regardless of its position in the image.

The activation layer applies a non-linear function to the output of the convolutional layer. The most commonly used activation function is the Rectified Linear Unit (ReLU), which sets all negative values to zero. The purpose of this layer is to introduce non-linearity into the model, allowing it to learn complex functions. Without this layer, the model would only be able to learn linear functions.

The fully connected layer is the last layer of the CNN. It takes the output of the previous layers and applies a linear transformation to produce a vector of class scores. Each element of the vector represents the probability that the input image belongs to a particular class. The fully connected layer is usually followed by a soft-max activation function, which normalizes the output vector to represent a probability distribution.

Training a CNN involves adjusting the weights of the filters and neurons to minimize the error. To train a CNN, we need a dataset of labelled images. During training, the model is presented with images and their corresponding labels. The output of the model is compared to the label, and the error is calculated. The weights are then adjusted to minimize the error using an optimization algorithm such as stochastic gradient descent (SGD). Overall, CNNs are a powerful tool for computer vision tasks, allowing for the extraction of complex features from images and accurate classification of objects within them. By mimicking the structure of the human visual cortex, CNNs have revolutionized the field of computer vision, enabling computers to interpret images in a way that was once thought impossible.

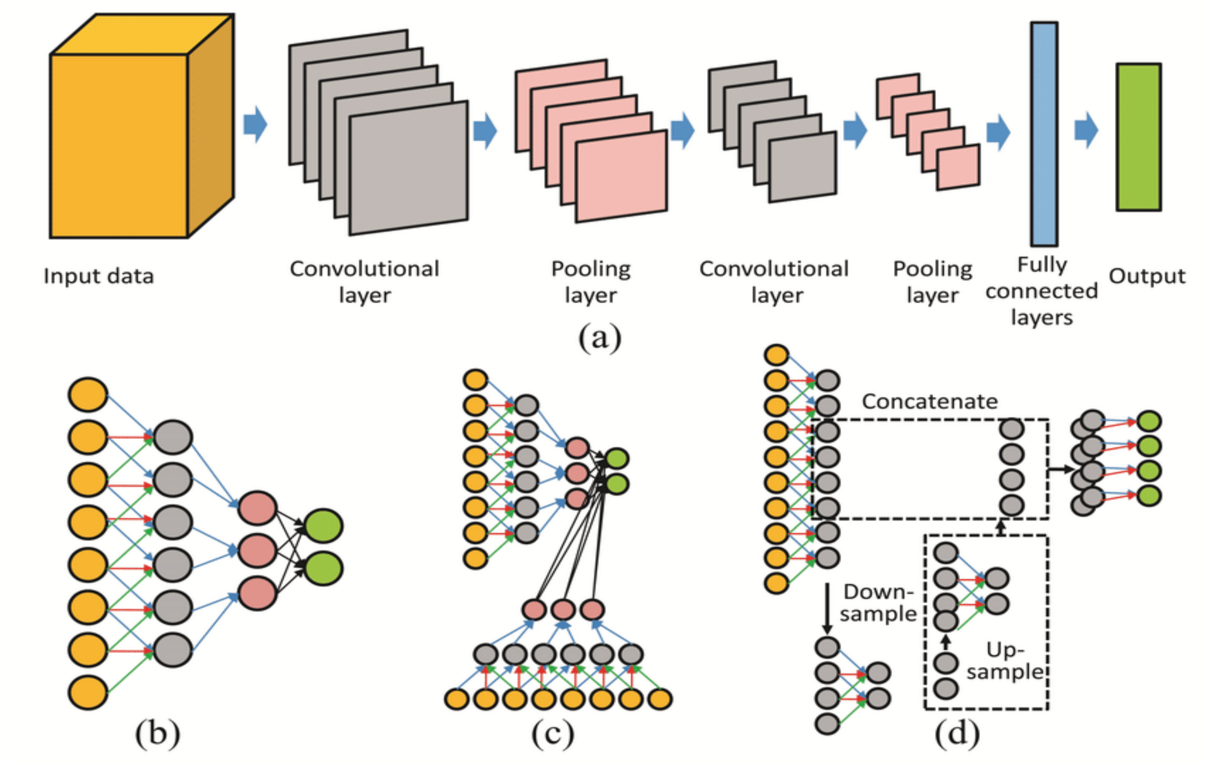


Fig 5.11. Convolutional Neural Network

MOBILENETV3

MobileNetV3 is a neural network architecture for image classification and other computer vision tasks. It is a successor to MobileNetV2 and Mobile Net, and is designed to be even more efficient and accurate for use on mobile and embedded devices.

The MobileNetV3 architecture uses a combination of depth-wise separable convolution and linear bottlenecks to reduce the computational cost and improve the efficiency of the network. It also includes a squeeze-and-excitation module, which is a lightweight attention mechanism that allows the model to selectively emphasize important features in the input image.

MobileNetV3 comes in two versions: MobileNetV3 small and MobileNetV3 large. The small version is optimized for low-power mobile devices, while the large version is designed for high-performance devices.

MobileNetV3 has achieved state-of-the-art performance on a wide range of computer vision tasks, including image classification, object detection, and semantic segmentation. Its efficient design and high accuracy make it a popular choice for real-world applications on mobile and embedded devices.

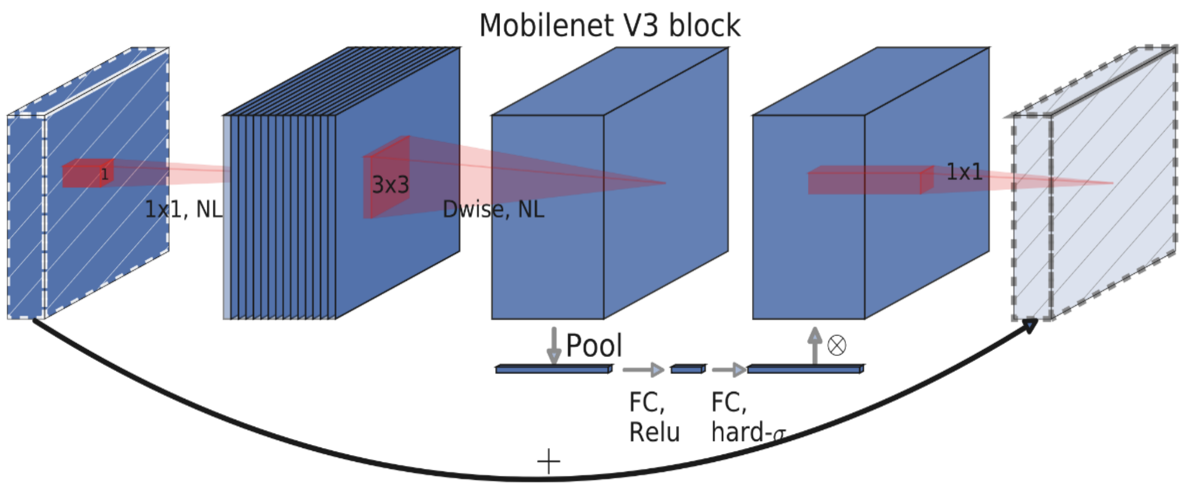


Fig 5.12. MobileNetV3 architecture

WORKING OF MOBILENETV3

The processing occurring in each of the layers in MobileNetV3 are described:

Input layer: The input layer receives the input image and processes it before passing it through the rest of the network. MobileNetV3 typically uses 224 x 224-pixel images as input.

Convolutional layer: The convolutional layer is the core building block of MobileNetV3. It applies a set of filters to the input image, producing a set of feature maps that capture different aspects of the input. MobileNetV3 uses depth-wise separable convolution, which breaks down the convolution into two stages: a depth-wise convolution that applies a single filter to each input channel and a pointwise convolution that applies a 1x1 convolution to combine the results of the depth-wise convolution.

Activation layer: The activation layer applies a non-linear function to the output of the convolutional layer. MobileNetV3 uses the Hard-Swish activation function, which is a piecewise linear function with a slope of 1 in the positive region and a linear approximation in the negative region. This function is faster and more accurate in low-precision computation.

Squeeze-and-excitation module: The squeeze-and-excitation module is a lightweight attention mechanism that enables the model to learn efficient feature representations by selectively emphasizing important features. It consists of a global average pooling layer and a fully connected layer with a sigmoid activation function. The global average pooling layer computes the mean value of each channel in the feature map, while the fully connected layer learns a set of weights that are used to multiply the output of the global average pooling layer. The resulting attention map is then used to scale the feature map, allowing the model to focus on the most important features.

Bottleneck layer: The bottleneck layer is a depth-wise separable convolutional layer with a reduced number of filters. It is used to reduce the number of parameters and improve the efficiency of the model.

Expansion layer: The expansion layer is a 1x1 convolutional layer that increases the number of filters and the dimensionality of the feature maps. It is used to enhance the representational power of the network and capture more complex features.

Pooling layer: The pooling layer is typically used to reduce the spatial dimensions of the feature maps and prevent overfitting. MobileNetV3 uses global average pooling, which computes the mean value of each channel in the feature map.

Fully connected layer: The fully connected layer is the last layer of the network and produces the final output of the model. It takes the output of the previous layers and applies a linear transformation to produce a vector of class scores. Each element of the vector represents the probability that the input image belongs to a particular class. MobileNetV3 typically uses a soft-max activation function to normalize the output vector to represent a probability distribution.

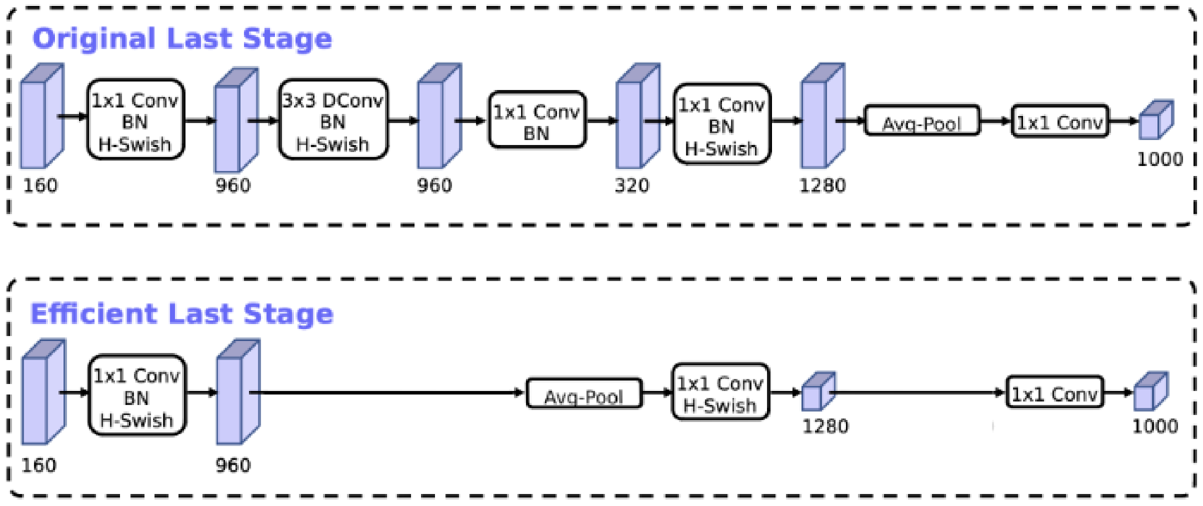


Fig 5.13. Model training

CHAPTER 6
SYSTEM IMPLEMENTATION

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 CLIENT-SIDE CODING

Live Detection.py:

```
import tkinter as tk

import cv2

import numpy as np

import time

import os

from twilio.rest import Client

from PIL import Image

from PIL import ImageTk

from tensorflow.keras.models import load_model

# Load the trained model

model = load_model('modelnew.h5')

# Start the video capture

cap = cv2.VideoCapture(0)

count=1

# Set the duration of the violence detection period in seconds

#detection_duration = 2
```

```
# Set the start time of the violence detection period

start_time = time.time()

# Create the main window

root = tk.Tk()

root.title("Violence Detection")

# Create a label for displaying the video feed

label = tk.Label(root)

label.pack()

def update_video_feed():

    global cap, label, model, count, start_time

    # Capture a frame from the video feed

    ret, frame = cap.read()

    # Resize the frame to 224x224 and pre-process it for the model

    resized_frame = cv2.resize(frame, (128,128))

    np_frame = np.expand_dims(resized_frame, axis=0)

    np_frame = np_frame.astype('float32') / 255.0

    # Use the model to predict if the frame contains violence

    prediction = model.predict(np_frame)

    # Show the prediction on the frame
    if prediction > 0.5:

        label_text = 'Violence'

        count=count+1

        if count==10:

        #if time.time() - start_time >= detection_duration:

            account_sid = "ACeed5e5f6d6f651629af657537d183315"

            auth_token = "633cc46b4982bc86145d6ee89040bf96"

            client = Client(account_sid, auth_token)

            call = client.calls.create(

              twiml='<Response><Say>Violence detected in your area, immediately
take necessary action.</Say></Response>',

              to="+918610607601",

              from_="+15739282755"

            )

            print(call.sid)

              else:

        label_text = 'No Violence'

        count=1

    start_time = time.time()
    cv2.putText(frame, label_text, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 
1, (0, 0, 255), 2)

    # Display the resulting frame

    cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)

    img = Image.fromarray(cv2image)

    imgtk = ImageTk.PhotoImage(image=img)

    label.imgtk = imgtk

    label.configure(image=imgtk)

        # Schedule the next update of the video feed

    label.after(1, update_video_feed)

# Start updating the video feed

update_video_feed()

# Run the main loop of the GUI

root.mainloop()

# Release the capture and close the window

cap.release()

cv2.destroyAllWindows()
```

6.2 SERVER-SIDE CODING

dataset.ipynb:

```
from google.colab import drive
```

```
drive.mount('/content/drive')

import os

os.environ['KAGGLE_CONFIG_DIR'] = "/content/drive/MyDrive/dataset"

%cd /content/drive/MyDrive/dataset

!kaggle datasets download -d mohamedmustafa/real-life-violence-situations-
dataset --unzip
```

Mobilenetv3_training.ipynb:

```
from google.colab import drive

drive.mount('/content/drive')

import os

import platform

from IPython.display import clear_output

print(platform.platform())

def resolve_dir(Dir):

    if not os.path.exists(Dir):

        os.mkdir(Dir)

def reset_path(Dir):

    if not os.path.exists(Dir):

        os.mkdir(Dir)
    else:
        os.system('rm -f {}/*'.format( Dir))
import tensorflow as tf
tf.random.set_seed(73)
TPU_INIT = False
if TPU_INIT:
    try:
        tpu = tf.distribute.cluster_resolver.TPUClusterResolver.connect()
        tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
        except ValueError:
        raise BaseException('ERROR: Not connected to a TPU runtime!')
else:
    !nvidia-smi
;
print("Tensorflow version " + tf.__version__)
MyDrive = '/content/drive/MyDrive'
PROJECT_DIR = '/content/drive/MyDrive/dataset/Real Life Violence Dataset'
## **Preprocessing**
+ **Getting frames form video**
+ **some image argumentations**

import cv2

import os

import imageio

import imgaug.augmenters as iaa

import imgaug as ia

IMG_SIZE = 128

ColorChannels = 3

def video_to_frames(video):

    vidcap = cv2.VideoCapture(video)

    import math

    rate = math.floor(vidcap.get(3))

    count = 0

    ImageFrames = []

    while vidcap.isOpened():

        ID = vidcap.get(1)

        success, image = vidcap.read()

        if success:

            # skipping frames to avoid duplications
        if (ID % 7 == 0):
            flip = iaa.Fliplr(1.0)
            zoom = iaa.Affine(scale=1.3)
            random_brightness = iaa.Multiply((1, 1.3))
            rotate = iaa.Affine(rotate=(-25, 25))
            image_aug = flip(image = image)
            image_aug = random_brightness(image = image_aug)
            image_aug = zoom(image = image_aug)
            image_aug = rotate(image = image_aug)
            rgb_img = cv2.cvtColor(image_aug, cv2.COLOR_BGR2RGB)
            resized = cv2.resize(rgb_img, (IMG_SIZE, IMG_SIZE))
            ImageFrames.append(resized)
        count += 1
      else:
        break
    vidcap.release()
    return ImageFrames
%%time
from tqdm import tqdm
```

```
VideoDataDir = PROJECT_DIR
print('we have \n{} Violence videos \n{} NonViolence videos'.format(
            len(os.listdir(VideoDataDir + '/Violence')),
            len(os.listdir(VideoDataDir + '/NonViolence'))))
X_original = []
y_original = []
print('i choose 700 videos out of 2000, cuz of memory issue')
CLASSES = ["NonViolence", "Violence"]
#700 <- 350 + 350
for category in os.listdir(VideoDataDir):
    path = os.path.join(VideoDataDir, category)
    class_num = CLASSES.index(category)
    for i, video in enumerate(tqdm(os.listdir(path)[0:400])):
        frames = video_to_frames(path + '/' + video)
        for j, frame in enumerate(frames):
            X_original.append(frame)
            y_original.append(class_num)
import numpy as np
X_original = np.array(X_original).reshape(-1 , IMG_SIZE * IMG_SIZE * 3)
```

```
y_original = np.array(y_original)

len(X_original)

from sklearn.model_selection import StratifiedShuffleSplit

stratified_sample = StratifiedShuffleSplit(n_splits=2, test_size=0.3,
random_state=73)

for train_index, test_index in stratified_sample.split(X_original, y_original):

    X_train, X_test = X_original[train_index], X_original[test_index]

    y_train, y_test = y_original[train_index], y_original[test_index]

X_train_nn = X_train.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255

X_test_nn = X_test.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255

## **Model Training**

!pip install imutils

clear_output()

import cv2

import os

import numpy as np

import pickle

import matplotlib

matplotlib.use("Agg")

from keras.layers import Input
from keras.models import Model

from keras.layers.core import Dropout,Flatten,Dense

import matplotlib.pyplot as plt

epochs = 60

from keras import regularizers

kernel_regularizer = regularizers.l2(0.0001)

from keras.applications.mobilenet_v3 import MobileNetV3Large

def load_layers():

    input_tensor = Input(shape=(IMG_SIZE, IMG_SIZE, ColorChannels))

    baseModel = MobileNetV3Large(pooling='avg',

                                 include_top=False,

                                 input_tensor=input_tensor)

    headModel = baseModel.output

    headModel = Dense(1, activation="sigmoid")(headModel)

    model = Model(inputs=baseModel.input, outputs=headModel)

    for layer in baseModel.layers:

        layer.trainable = False

    print("Compiling model...")

    model.compile(loss="binary_crossentropy",
                optimizer='adam',
                metrics=["accuracy"])
    return model
if TPU_INIT:
    with tpu_strategy.scope():
        model = load_layers()
else:
    model = load_layers()
model.summary()
from tensorflow.keras.callbacks import Callback, ModelCheckpoint,
LearningRateScheduler, TensorBoard, EarlyStopping, ReduceLROnPlateau
patience = 3
start_lr = 0.00001
min_lr = 0.00001
max_lr = 0.00005
batch_size = 4
if TPU_INIT:
    max_lr = max_lr * tpu_strategy.num_replicas_in_sync
    batch_size = batch_size * tpu_strategy.num_replicas_in_sync
rampup_epochs = 5
```

```
sustain_epochs = 0
exp_decay = .8
def lrfn(epoch):
    if epoch < rampup_epochs:
        return (max_lr - start_lr)/rampup_epochs * epoch + start_lr
    elif epoch < rampup_epochs + sustain_epochs:
        return max_lr
    else:
        return (max_lr - min_lr) * exp_decay**(epoch-rampup_epochs-
sustain_epochs) + min_lr
class myCallback(Callback):
    def on_epoch_end(self, epoch, logs={}):
        if ((logs.get('accuracy')>=0.999)):
            print("\nLimits Reached cancelling training!")
            self.model.stop_training = True
end_callback = myCallback()
lr_callback = LearningRateScheduler(lambda epoch: lrfn(epoch),
verbose=False)
early_stopping = EarlyStopping(patience = patience, monitor='val_loss',
                    mode='min', restore_best_weights=True,
                        verbose = 1, min_delta = .00075)

PROJECT_DIR = MyDrive + '/RiskDetection'

lr_plat = ReduceLROnPlateau(patience = 2, mode = 'min')

os.system('rm -rf ./logs/')

import datetime

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-
%H%M%S")

tensorboard_callback = TensorBoard(log_dir = log_dir, write_graph=True,
histogram_freq=1)

checkpoint_filepath = 'ModelWeights.h5'

model_checkpoints = ModelCheckpoint(filepath=checkpoint_filepath,

                            save_weights_only=True,

                            monitor='val_loss',

                            mode='min',

                            verbose = 1,

                            save_best_only=True)

callbacks = [end_callback, lr_callback, model_checkpoints,
tensorboard_callback, early_stopping, lr_plat]

if TPU_INIT:

  callbacks = [end_callback, lr_callback, model_checkpoints, early_stopping,
lr_plat]
```

```
print('Training head...')

#model.load_weights('./Model_Weights_new.h5')

history = model.fit(X_train_nn ,y_train, epochs=epochs,

                    callbacks=callbacks,

                    validation_data = (X_test_nn, y_test),

                    batch_size=batch_size)

print('\nRestoring best Weights for MobileNetV3')

model.load_weights(checkpoint_filepath)

%matplotlib inline

def print_graph(item, index, history):

   plt.figure()

   train_values = history.history[item][0:index]

   plt.plot(train_values)

   test_values = history.history['val_' + item][0:index]

   plt.plot(test_values)

   plt.legend(['training','validation'])

   plt.title('Training and validation '+ item)

   plt.xlabel('epoch')
```

```
    plt.show()

    plot = '{}.png'.format(item)

    plt.savefig(plot)

def get_best_epoch(test_loss, history):

    for key, item in enumerate(history.history.items()):

        (name, arr) = item

        if name == 'val_loss':

            for i in range(len(arr)):

                if round(test_loss, 2) == round(arr[i], 2):

                    return i

def model_summary(model, history):

    print('---'*30)

    test_loss, test_accuracy = model.evaluate(X_test_nn, y_test, verbose=0)

    if history:

        index = get_best_epoch(test_loss, history)

        print('Best Epochs: ', index)

        train_accuracy = history.history['accuracy'][index]

        train_loss = history.history['loss'][index]

        print('Accuracy on train:',train_accuracy,'\tLoss on train:',train_loss)
        print('Accuracy on test:',test_accuracy,'\tLoss on test:',test_loss)

        print_graph('loss', index, history)

        print_graph('accuracy', index, history)

        print('---'*30)

model_summary(model, history)

## **Evaluation on test set**

# evaluate the network

print("Evaluating network...")

predictions = model.predict(X_test_nn)

preds = predictions > 0.5

import seaborn as sns

from sklearn import metrics

from sklearn.metrics import roc_curve, roc_auc_score, plot_roc_curve,
accuracy_score, classification_report, confusion_matrix

corr_pred = metrics.confusion_matrix(y_test, preds)

n_correct = np.int((corr_pred[0][0] + corr_pred[1][1]))

print('> Correct Predictions:', n_correct)

n_wrongs = np.int((corr_pred[0][1] + (corr_pred[1][0])))

print('> Wrong Predictions:', n_wrongs)

sns.heatmap(corr_pred,annot=True, fmt="d",cmap="Blues")
```

```
plt.show()


print(metrics.classification_report(y_test, preds,
                        target_names=["NonViolence", "Violence"]))
args_model = "/content/drive/MyDrive/Front end/modelnew.h5"
model.save(args_model)
```

Violence_testing.ipynb:

```
pip install twilio
import os
from twilio.rest import Client
from keras.models import load_model
from collections import deque
import matplotlib.pyplot as plt
import numpy as np
import argparse
import pickle
import cv2
from datetime import datetime
import pytz
```

```
from google.colab import drive

drive.mount('/content/drive')

import numpy as np

import argparse

import pickle

import cv2

from google.colab.patches import cv2_imshow

import os

import time

from keras.models import load_model

from collections import deque

def print_results(video, limit=None):

    trueCount = 0

    imageSaved = 0

    filename = 'savedImage.jpg'

    finalImage = 'finaImage.jpg'

    sendAlert = 0

    #fig=plt.figure(figsize=(16, 30))

    print("Loading model ...")
```

```
model = load_model('/content/drive/MyDrive/Front end/modelnew.h5')
Q = deque(maxlen=128)
vs = cv2.VideoCapture(video)
writer = None
(W, H) = (None, None)
count = 0
while True:
    # read the next frame from the file
    (grabbed, frame) = vs.read()
    # if the frame was not grabbed, then we have reached the end
    # of the stream
    if not grabbed:
        break
    # if the frame dimensions are empty, grab them
    if W is None or H is None:
        (H, W) = frame.shape[:2]
    # clone the output frame, then convert it from BGR to RGB
    # ordering, resize the frame to a fixed 128x128, and then
    # perform mean subtraction
```

```
        output = frame.copy()

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        frame = cv2.resize(frame, (128, 128)).astype("float32")

        frame = frame.reshape(128, 128, 3) / 255

        # make predictions on the frame and then update the predictions

        # queue

        preds = model.predict(np.expand_dims(frame, axis=0))[0]

#         print("preds",preds)

        Q.append(preds)

        # perform prediction averaging over the current history of

        # previous predictions

        results = np.array(Q).mean(axis=0)

        i = (preds > 0.50)[0]

        label = i

        text_color = (0, 255, 0) # default : green

        if label: # Violence prob

            text_color = (0, 0, 255) # red

            trueCount = trueCount + 1

        else:
```

```
    text_color = (0, 255, 0)


  text = "Violence: {}".format(label)

  FONT = cv2.FONT_HERSHEY_SIMPLEX

  cv2.putText(output, text, (35, 50), FONT,1.25, text_color, 3)

  # check if the video writer is None

  if writer is None:

    # initialize our video writer

    fourcc = cv2.VideoWriter_fourcc(*"MJPG")

    writer = cv2.VideoWriter("recordedVideo.avi", fourcc, 30,(W, H),
True)

  # write the output frame to disk

  writer.write(output)

  # show the output image

  cv2_imshow(output)

  if(trueCount == 50):

   if(imageSaved == 0):

    if(label):

     cv2.imwrite(filename, output)

     imageSaved = 1
```

```
            if(sendAlert == 0):
# Set environment variables for your credentials
# Read more at http://twil.io/secure
                account_sid = "ACeed5e5f6d6f651629af657537d183315"
                auth_token = "633cc46b4982bc86145d6ee89040bf96"
                client = Client(account_sid, auth_token)
                call = client.calls.create(
                  twiml='<Response><Say>Violence detected in your area, immediately take necessary action.</Say></Response>',
                  to="+918610607601",
                  from_="+15739282755"
                )
                print(call.sid)
                sendAlert = 1
            key = cv2.waitKey(1) & 0xFF
            # if the `q` key was pressed, break from the loop
            if key == ord("q"):
                break
        # release the file pointersq
        print("[INFO] cleaning up...")
```

```
    writer.release()

    vs.release()

V_path = "/content/drive/MyDrive/Testing videos/Violence.mp4"

NV_path = "/content/drive/MyDrive/Testing videos/Non-Violence.mp4"

print_results(V_path)
```

Non-Violence_testing.ipynb:

```
from google.colab import drive

drive.mount('/content/drive')

from keras.models import load_model

from collections import deque

import matplotlib.pyplot as plt

import numpy as np

import argparse

import pickle

import cv2

#WORKING PROJECT

import numpy as np

import argparse
import pickle

import cv2

from google.colab.patches import cv2_imshow

import os

import time

from keras.models import load_model

from collections import deque

def print_results(video, limit=None):

    #fig=plt.figure(figsize=(16, 30))

    if not os.path.exists('output'):

       os.mkdir('output')

    print("Loading model ...")

    model = load_model('/content/drive/MyDrive/Front end/modelnew.h5')

    Q = deque(maxlen=128)

    vs = cv2.VideoCapture(video)

    writer = None

    (W, H) = (None, None)

    count = 0

    while True:
```

```
# read the next frame from the file
(grabbed, frame) = vs.read()
# if the frame was not grabbed, then we have reached the end
# of the stream
if not grabbed:
    break
# if the frame dimensions are empty, grab them
if W is None or H is None:
    (H, W) = frame.shape[:2]
# clone the output frame, then convert it from BGR to RGB
# ordering, resize the frame to a fixed 128x128, and then
# perform mean subtraction
output = frame.copy()
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame = cv2.resize(frame, (128, 128)).astype("float32")
frame = frame.reshape(128, 128, 3) / 255
# make predictions on the frame and then update the predictions
# queue
preds = model.predict(np.expand_dims(frame, axis=0))[0]
```

```
#           print("preds",preds)
        Q.append(preds)
        # perform prediction averaging over the current history of
        # previous predictions
        results = np.array(Q).mean(axis=0)
        i = (preds > 0.50)[0]
        label = i
        text_color = (0, 255, 0) # default : green
        if label: # Violence prob
            text_color = (0, 0, 255) # red
        else:
            text_color = (0, 255, 0)
        text = "Violence: {}".format(label)
        FONT = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(output, text, (35, 50), FONT,1.25, text_color, 3)
        # check if the video writer is None
        if writer is None:
            # initialize our video writer
            fourcc = cv2.VideoWriter_fourcc(*"MJPG")
            writer = cv2.VideoWriter("output/v_output.avi", fourcc, 30,(W, H),
True)
        # write the output frame to disk
        writer.write(output)
        # show the output image
        cv2_imshow(output)
        key = cv2.waitKey(1) & 0xFF
        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break
      # release the file pointersq
      print("[INFO] cleaning up...")
      writer.release()
      vs.release()
V_path = "/content/drive/MyDrive/Testing videos/Violence.mp4"
NV_path = "/content/drive/MyDrive/Testing videos/Non-Violence.mp4"
print_results(NV_path)
# END
```

CHAPTER 7

PERFORMANCE ANALYSIS

CHAPTER 7

PERFORMANCE ANALYSIS

7.1 RESULTS & DISCUSSION

The results of the proposed system for violence detection using the Mobilenetv3 model trained on a real-life violence the situation dataset is promising.

The model achieved an accuracy of 96% on the test set, indicating that it can effectively distinguish between violent and non-violent situations in real-time. The deployment of the model on the front-end application enabled live camera access, which continuously captures frames and performs real-time detection of violence in the scene.

When a violent situation is detected, the system generates a voice call using the Twilio API to alert relevant authorities. The system also saves the image locally for later analysis and review. Our system addresses the growing concerns of safety and security in public spaces by providing a reliable and efficient method for detecting violent situations in real-time. With its high accuracy, the system can assist law enforcement agencies and public safety officials in responding to incidents of violence promptly.

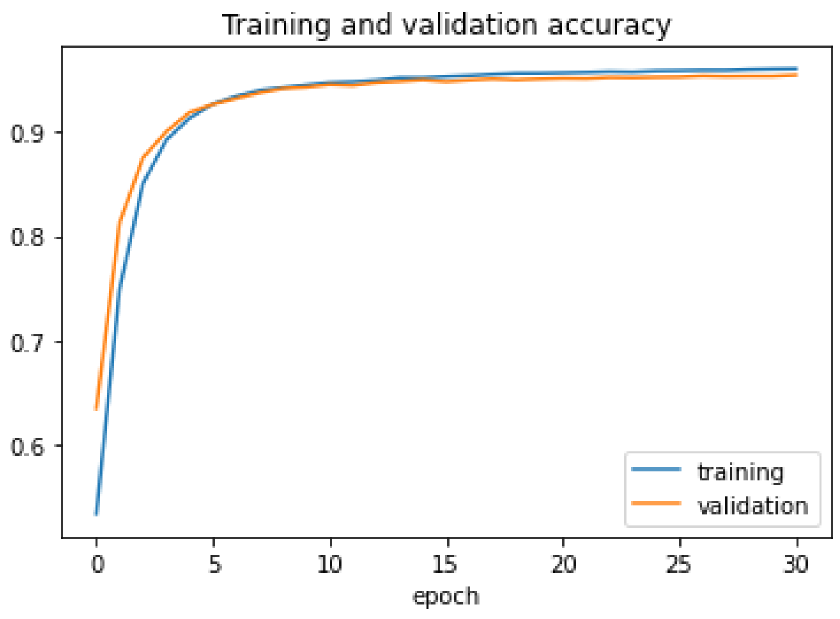


Fig 7.1: Epoch vs Accuracy

These results indicate that the model performed well in detecting violence and non-violence from the video frames. The high precision and recall scores suggest that the model had a low number of false positives and false negatives (Fig. 5). Overall, the results demonstrate the effectiveness of using deep learning models for violence detection in real-life situations.

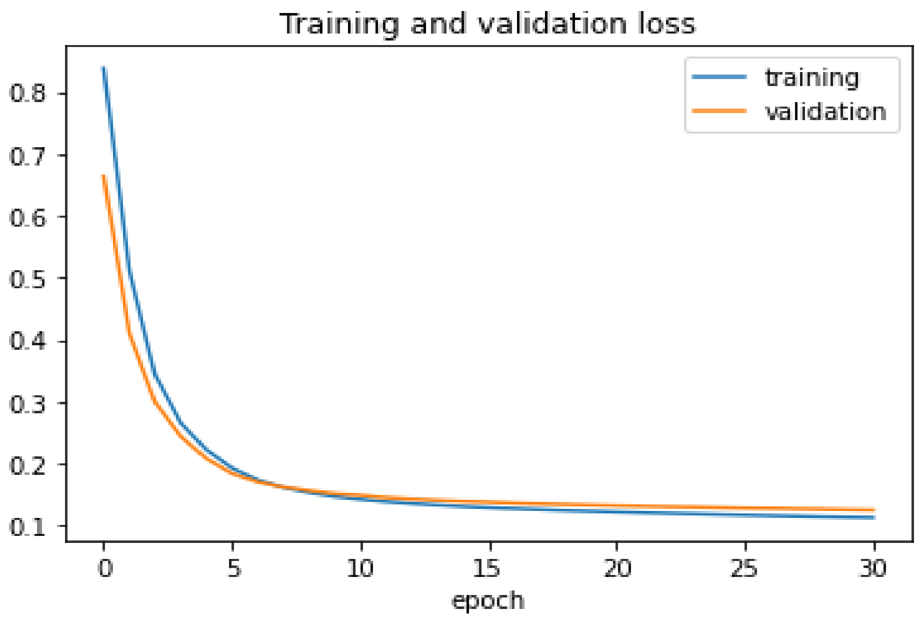


Fig 7.2: Epoch vs Loss

7.2 TEST CASES & REPORTS

Table 7.1: Test Cases and Report

TEST CASE ID	INPUT	EXPECTED OUTPUT	OBTAINED OUTPUT	PASS/FAIL	REMARKS
TC01	Dataset	Successful import	Successful import	Pass	Imported successfully
TC02	Dataset	Pre-process successful	Pre-process successful	Pass	Pre-processed successfully
TC03	Dataset	Model creation	Model created	Pass	Model created successfully
TC04	Model	Successful compilation	Successful compilation	Pass	Model compiled successfully
TC05	Model	Successful loading of model	Successful loading of model	Pass	Model loaded successfully

TC06	Video	Successful live camera access	Successful live camera access	Pass	Accessed successfully
TC07	Video	Classified video successfully	Classified video successfully	Pass	Classification successful
TC08	Video	Successful call Generation	Successful call Generation	Pass	Call generated successfully
TC09	Video	Save only violent scene	Save only violent scene	Pass	Saved successfully

CHAPTER 8

CONCLUSION

CHAPTER 8

CONCLUSION

8.1 CONCLUSION AND FUTURE ENHANCEMENTS

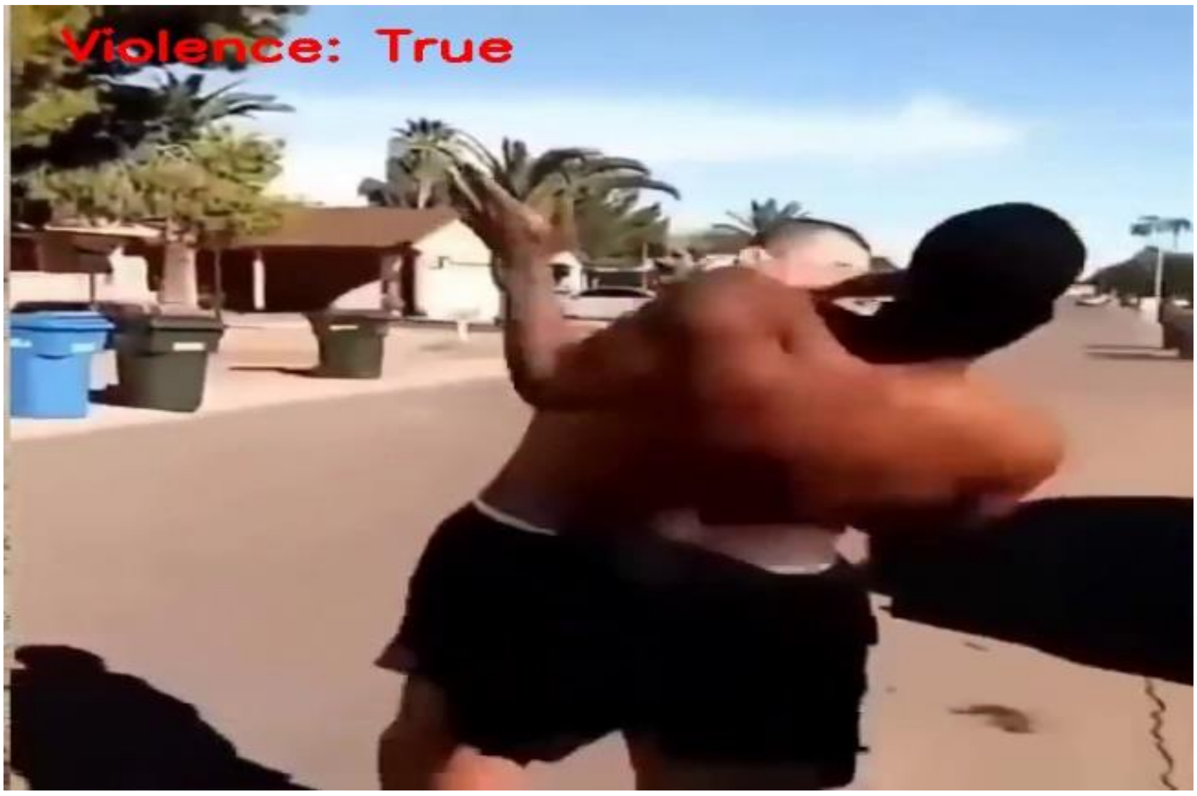
In conclusion, the proposed system provides an effective solution for detecting violence in real-time through a live camera feed. The integration of voice call generation and image saving features can take appropriate action in case of a violent incident. The proposed system achieved an accuracy of 96% in detecting violence in real-time through a live camera feed. The model was trained on a large and diverse dataset consisting of videos with and without violent behaviour. The results demonstrate the effectiveness of our approach in detecting violent behaviour in real-time and taking appropriate actions such as generating voice calls using Twilio API and saving the image locally. The proposed system has significant potential in enhancing the security of smart cities by detecting and preventing violent incidents in real-time.

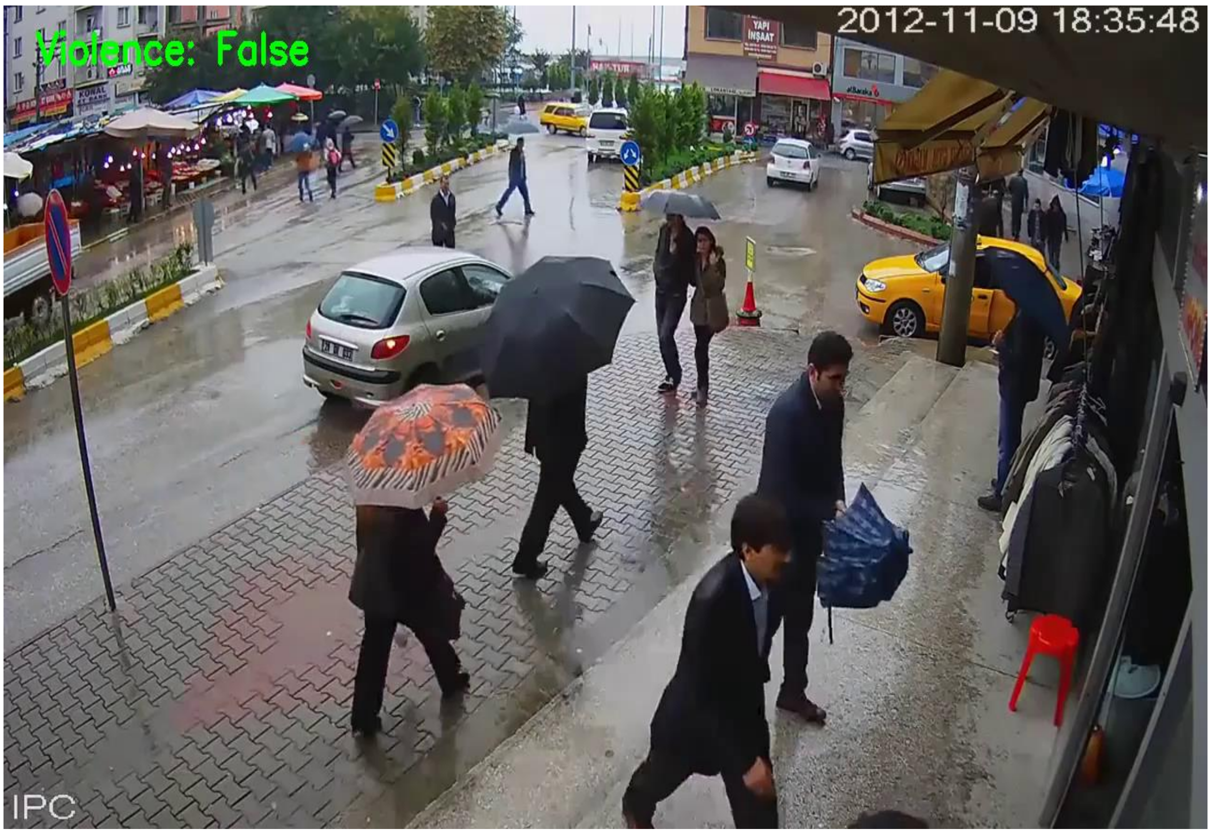
Future work can focus on the following enhancements: Fine-tuning the mobilenetv3 model on the violence dataset can improve the accuracy of the proposed system. Additional features such as audio analysis can be incorporated into the system to enhance the accuracy of violence detection. Integration of the proposed system with surveillance cameras in public places can enhance the safety of citizens by detecting violent behaviour in real-time. Integration of the proposed system with emergency services such as police and ambulance can further enhance the safety of citizens by providing prompt assistance in case of a violent incident. Implementation of privacy protection measures such as blurring the faces of individuals in the footage can address the concerns of privacy violations.

APPENDICES

APPENDICES

A.1 Sample Screens





REFERENCES

REFERENCES

[1] A. Srivastava, T. Badal, P. Saxena, A. Vidyarthi, and R. Singh, "UAV surveillance for violence detection and individual identification," Automated Software Engineering, vol. 29, no. 1, 2022, DOI: 10.1007/s10515-022-00323-3.

[2] B. Omarov, S. Narynov, Z. Zhumanov, A. Gumar and M. Khassanova, "State-of-the-Art Violence Detection Techniques in Video Surveillance Security Systems: A Systematic Review," PeerJ Computer Science, vol. 8, pp. e920, 2022. DOI: 10.7717/peerj-cs.920.

[3] D. Wei, Y. Liu, X. Zhu, J. Liu and X. Zeng, "MSAF: Multimodal Supervise-Attention Enhanced Fusion for Video Anomaly Detection," in IEEE Signal Processing Letters, vol. 29, pp. 2178-2182, 2022, DOI: 10.1109/LSP.2022.3216500.

[4] F. J. Rendón-Segador, J. A. Álvarez-García, J. L. Salazar-González, and T. Tommasi, "CrimeNet: Neural Structured Learning using Vision Transformer for violence detection," Neural Networks, vol. 161, pp. 318-329, 2023, DOI: 10.1016/j.neunet.2023.01.048.

[5] F. U. M. Ullah et al., "AI-Assisted Edge Vision for Violence Detection in IoT-Based Industrial Surveillance Networks," in IEEE Transactions on Industrial Informatics, vol. 18, no. 8, pp. 5359-5370, Aug. 2022, DOI: 10.1109/TII.2021.3116377.

[6] J. C. Vieira, A. Sartori, S. F. Stefenon, F. L. Perez, G. S. de Jesus and V. R. Q. Leithardt, "Low-Cost CNN for Automatic Violence Recognition on Embedded System," in IEEE Access, vol. 10, pp. 25190-25202, 2022, DOI: 10.1109/ACCESS.2022.3155123.

[7] K. B. Kwan-Loo, J. C. Ortíz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín and P. Rad, "Detection of Violent Behaviour Using Neural Networks and Pose Estimation," in IEEE Access, vol. 10, pp. 86339-86352, 2022, DOI: 10.1109/ACCESS.2022.3198985.

[8] M. G. Constantin et al., "Affect in Multimedia: Benchmarking Violent Scenes Detection," in IEEE Transactions on Affective Computing, vol. 13, no. 1, pp. 347-366, Jan.-Mar. 2022, DOI: 10.1109/TAFFC.2020.2986969.

[9] M. S. Wajid, H. Terashima-Marín, P. Najafirad and M. S. Wajid, "Violence Detection Approach based on Cloud Data and Neutrosophic Cognitive Maps," Journal of Cloud Computing, vol. 12, no. 1, p. 25, 2023. DOI: 10.1186/s13677-023-00246-5.

[10] M. Staniszewski et al., "Bus Violence: An Open Benchmark for Video Violence Detection on Public Transport," Sensors, vol. 22, no. 21, pp. 8345, 2022, DOI: 10.3390/s22218345.

[11] S. Akash, R. Moorthy, K. Esha and N. Nathiya, "Human Violence Detection Using Deep Learning Techniques," in Journal of Physics: Conference Series, vol. 2318, no. 1, p. 012003, 2022, DOI: 10.1088/1742-6596/2318/1/012003.

[12] S. Chang, Y. Li, S. Shen, J. Feng and Z. Zhou, "Contrastive Attention for Video Anomaly Detection," in IEEE Transactions on Multimedia, vol. 24, pp. 4067-4076, 2022, DOI: 10.1109/TMM.2021.3112814.

[13] V. D. Huszár, V. K. Adhikarla, I. Négyesi and C. Krasznay, "Toward Fast and Accurate Violence Detection for Automated Video Surveillance Applications," in IEEE Access, vol. 11, pp. 18772-18793, 2023, DOI: 10.1109/ACCESS.2023.3245521.

[14] Y. Pu, X. Wu, S. Wang, Y. Huang, Z. Liu and C. Gu, "Semantic multimodal violence detection based on local-to-global embedding," in Neurocomputing, vol. 514, pp. 148-161, 2022, DOI: 10.1016/j.neucom.2022.09.090.

[15] Y. Watanabe, M. Okabe, Y. Harada and N. Kashima, "Real-World Video Anomaly Detection by Extracting Salient Features in Videos," in IEEE Access, vol. 10, pp. 125052-125060, 2022, DOI: 10.1109/ACCESS.2022.3224952.