Real Time Action Recognition from Video Footage

by

Tasnim Sakib Apon
20241068
Mushfiqul Islam Chowdhury
17101120
MD Zubair Reza
17101275
Arpita Datta
18341008
Syeda Tanjina Hasan
17101184

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
June 2021

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Tasnim Sakib Apon

Tasnim Sakib Apon
20241068

Mushfiq

Mushfiqul Islam Chowdhury
17101120

Zubair Reza

MD Zubair Reza
17101275

Arpita Datta

Arpita Datta
18341008

Syeda Tanjina Hasan

Syeda Tanjina Hasan
17101184

Approval

The thesis titled "Real Time Action Recognition from Video Footage" submitted by

1. Tasnim Sakib Apon (20241068)
2. Mushfiqul Islam Chowdhury (17101120)
3. MD Zubair Reza (17101275)
4. Arpita Datta (18341008)
5. Syeda Tanjina Hasan (17101184)

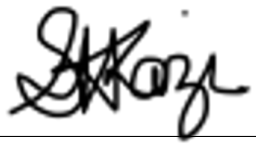
Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on june 10, 2021.

Examining Committee:

Supervisor:
(Member)

MD. GOLAM RABIUL ALAM
Associate Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)



Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

We, officially declare that the work is based on the results we received through our effort. All additional sources of information have been acknowledged in the text. No person has ever submitted this thesis, in whole or in part, to any other institution.

Abstract

Crime rate is increasing proportionally with the increasing rate of the population. The most prominent approach was to introduce Closed-Circuit Television (CCTV) camera-based surveillance to tackle the issue. Video surveillance cameras have added a new dimension to detect crime. Several research works on autonomous security camera surveillance are currently ongoing, where the fundamental goal is to discover violent activity from video feeds. From the technical viewpoint, this is a challenging problem because analyzing a set of frames, i.e., videos in temporal dimension to detect violence might need careful machine learning model training to reduce false results. This research focused on this problem by integrating state-of-the-art Deep Learning methods to ensure a robust pipeline for autonomous surveillance for detecting violent activities, e.g., kicking, punching, and slapping. Initially, we designed a dataset of this specific interest, which were 600 videos (200 for each action). Later, we have utilized existing pre-trained model architectures to extract features, followed by classification and accuracy analysis.Also, We have classified our models' accuracy, confusion matrix on different pre-trained architectures like VGG16, InceptionV3, ResNet50 and MobileNet V2. Among the pre-trained models VGG16 and MobileNet V2 performed better.

Keywords: Deep Neural Network, Deep learning, Real Time Action, Action Detection from Footage, Crime Detection from Footage, Surveillance action detection.

Dedication

Every challenging work requires self-effort as well as encouragement from the elders, particularly those who were very close to our hearts. We devote our humble efforts to our caring parents, whose affection, devotion, motivation and prayer day and night make us worthy of such achievement and honor, along with all the hard-working and respected Teachers.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.

Secondly, to our advisor Dr. Md. Golam Rabiul Alam sir, our supervisor who has constantly aided us in our efforts and has been by our side through the thick and thins of this work and continuously encouraged us to complete our work in time. We are grateful to have been able to work under his supervision.

Finally, Our gratitude also extends towards BRAC University, and the faculty members of the Department of Computer Science and Engineering, from whom we have gained the very knowledge and experience we needed to complete our thesis.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

ANN Artificial Neural Network

$CIFAR$ Canadian Institute For Advanced Research

CNN Convolutional Neural Network

$DCNN$ Deep Convolutional Neural Network

DNN Deep Neural Network

HNF Hybrid N-feature

HOG Histogram of Oriented Gradients

KNN K-Nearest Neighbor

$MoSIFT$ Motion SIFT

$ReLU$ Rectified Linear Unit

RGB Red, Green, Blue

$SCFL$ Semi-Supervised Cross Feature Learning

$STIP$ Spatio-Temporal Interest Point

SVM Support Vector Machine

VGG Visual Geometry Group

Chapter 1

Introduction

Recently, with the growth of the population, the crime rate is increasing day by day, and physical bullying or torturing are also happening with anyone. Many crimes are happening, and it is not under the control of the police or CID or security board. The rate of bullying, harassment, or torturing has increased a lot and is happening almost every single time, and so the crimes are also increasing. These situations are not under control fully. In the past, technologies have been used which were not that effective to detect violence or bullying. Video surveillance gives a good part in real-time action recognition. Cameras are formed at each corner since the video surveillance system recognizes the scenes and detects divergent actions. Video crime detection is an essential subject in computer vision. Nowadays, from home to street, from person to crowd, the percentage of violent activities such as physical bullying has increased dramatically. In nearly every industry, surveillance cameras are used. Consequently, due to the inefficiency of recordings, human monitoring on surveillance cameras is becoming redundant. These recordings are never seen in most cases all time long in a day and the importance of these cameras is reducing for that reason. Computer interference in management will substantially eliminate the problem of inactivity. It has become an important subject to make machines understand videos' violent actions to imbrute the process. This paper introduces a novel technique on this subject and effectively enhances the quality of violent physical bullying video classification. The idea of violent action, defines physical bullying which means harming any person by affecting his/her body. For violent action identification (e.g., physical bullying), not much research has been completed. Our subsidies in this paper are, here, we will present the new datasets of physical bullying of fighting, slapping, punching, containing real-world fights. After that, we will propose embryonic procedures to intercept the fight detection problem, which will suffice as a touchstone for upcoming experimentation in the domain.

1.1 Motivation

In the current situation we can observe that accidents are increasing exponentially in our country, Bangladesh. There are so many incidents that people are facing in their daily life. For example, Robbing, Stealing, road accidents, breaking rules, careless driving, medical emergencies and what not? Generally, we want to know the real time of that action so that we can respond in an efficient way. To take care of these issues we planned to build a system that can help to predict real action

time from video footage. In Bangladesh people use different social media platforms like Facebook, Youtube etc, to upload videos from their daily life. Resulting in a huge database of video footage. This huge amount of data needs to be analyzed efficiently so that it could give us the accurate result. By applying Action recognition systems to these analyzed data we can identify the real-time when actually the action happened. Action recognition systems enable video surveillance cameras to be analyzed 24 hours a day, seven days a week, and an alarm will be activated in the event of a dangerous situation. The action has been recognized and, It has been possible only because of developing action recognition algorithms.

1.2 Problem Statement

Physical violence is a form of aggressive behavior in which someone intentionally and repeatedly causes another person physically injured. Physical violence is more than a fight, it does not only harm a person physically but also mentally. Physical abuse can occur anywhere. It can take place in your neighborhood, whereas reaching to school, at school, and even at your home. There is no foolproof solution of this. To solve this kind of problem many countries employ surveillance cameras. But it is not possible to monitor every surveillance camera every moment. Exactly where our research takes place. Imagine a tall strong man punching a helpless boy or a wicked boy slapping a girl. The issue is as terrible as it seems, and it will only get worse if we do nothing. The helpless boy or girl might be saved from physical bullying if our technology is placed in nearby CCTV cameras which detect real time action. There could be a street fight, physical bullying, or physical violence. Our system detects the three major types of violent behavior in real time. However, researching with Deep neural networks is always difficult since it requires a large dataset for training. We gathered data on three primary actions: slap, punch, and kick, and used several CNN architectures for real time action detection.

1.3 Aims and Objectives

This research aims to develop a model for detecting movement of fighting or physical bullying like slapping, kicking, punching etc from video footage. The objectives of this research are:

- To deeply understand crime or fighting scenes detection techniques.
- Implementing deep learning methods to recognise action form dataset.
- Applying the basic architectures like InceptionV3, VGG16, ResNet50 to get better output from our created model.
- Showing a brief comparison between pre-trained models.
- Analyzing the results of various models and finding the model with maximum accuracy to find out the best model to predict.

1.4 Contribution and Impact

Our work comprises a detailed evaluation of autonomous violent activity detection based on videos. We have collected a big dataset to design the pipeline, which can be extended in future works via augmentation or adding new violent action classes. Also, our detailed analysis of the existing pre-trained models may lead future research on this domain with a better understanding of the feature extractors and their impact on the real-world data. As we already know from this work that the deep learning models are data-hungry, several initiatives can be taken to capture more violent activities from real-world CCTV footage. Because, once a successful and robust training model is built, capturing the violent actions will become real-time and error-prone, which will broadly impact the police force of respective countries. Also, our approach will positively impact user privacy-preserving because a machine will execute the whole pipeline of surveillance. With that implication, we are now planning to increase the data more and evaluate more sophisticated approaches regarding this problem.

1.5 Thesis Overview

Chapter 1 is the introduction of our thesis work. Our motivations and objectives to do this thesis are also mentioned here as well as a short overview of the methodology that has been followed by us.

Chapter 2 consists of the literature review where we have demonstrated the background study that we have done for this thesis.

Chapter 3 consists of workflow, pre-processing and data labelling and pre-trained models for our thesis.

Chapter 4 consists of implementation of our overall thesis.

Chapter 5 contains the conclusion and future work.

Chapter 2

Literature Review

2.1 Deep Neural Network

A neural network is a network structure whose main purpose is to explain the relationships in a specified dataset using a mechanism similar to the human brain. A deep neural network is a part of neural network which has more than two layers. Deep neural networks produce data in a variety of ways using advanced statistical modeling. Deep neural networks need at least three layers: an input layer in the beginning , an output layer in the end, and in between input and output layers there is at least one unknown layer defined as the hidden layer. Each layer is responsible for a particular task. To create a feature hierarchy, each layer performs its own sorting. Acting with unlabeled or unstructured data is one of the major uses of these deep neural networks. A deep neural network with several hidden layers may conduct functions that would need a very large number of units per layer in a shallow network [3]. People use the word deep learning to describe deep neural networks. Deep learning is a subset of neural networks in which artificial intelligence-based technologies try to differentiate.

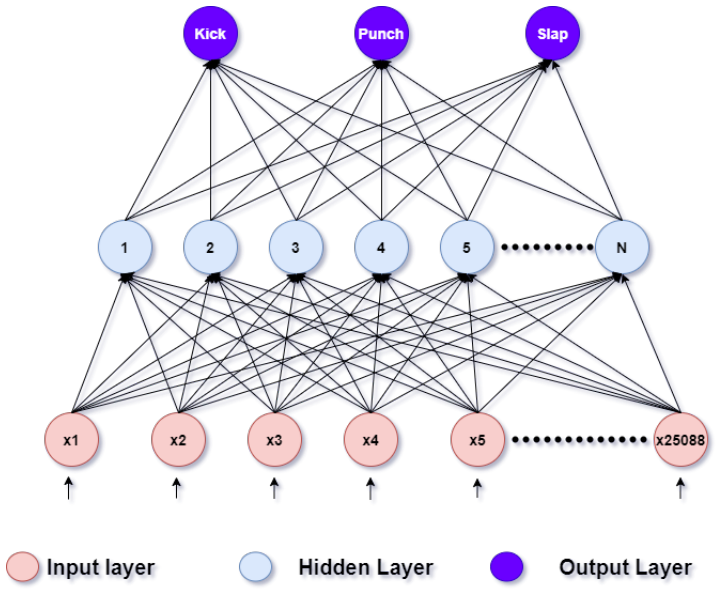


Figure 2.1: Deep neural network

There are many familiar single hidden layer neural networks but deep neural net-

works are distinct by their depth. In figure 2.1 there are three layers, Following the previous layer's performance or output, each layer of neurons trains on a separate set of characteristics.

2.1.1 Input Layer

Our input layer takes the output of pre-trained models. Outputs of pre-trained models are three dimensional values as we are working on image data which is converted from the action video dataset. This output is a matrix of pixel values. If we get an output of a pre-trained value (620, 8, 8, 512) then the input layer will hold 620 input image data with the value of (width=8, height=8, depth=512). Here Depth indicates RGB values. We have also normalized the pixel value to keep it between 0 to 1. To work with an input layer, we need to reshape this three dimensional value into one dimensional value. So, we have converted this value(width=8, height=8, depth=512) into one dimensional shape and created the input layer with this input shape. This input shape varies from pre-trained model to model. For VGG16 output shape is (Dataset Length, 7, 7, 512) so the converted input shape is 25088, InceptionV3's output shape is (Dataset Length, 5, 5, 2048) so the converted input shape is 51200 and ResNet50's output shape is (Dataset Length, 7, 7, 2048) so the converted input shape is 100352.

2.1.2 Rectified Linear Unit

In deep neural networks, ReLU or rectified linear unit is a non-linear activation element. The key benefit of utilizing the ReLU function over other activation functions is that it does not activate all the neurons during the same period of time. The Mathematical representation of ReLU is

$$f(x) = \max(0, x) \tag{2.1}$$

From the equation we can say ReLU output is the total length between predicted values of x and 0. When the input value is less than 0, the output equals 0, and when the input value is greater than 0, the output equals the input value. Since the derivative of the ReLU function is positive for a positive input, it will speed up deep neural network training relative to other known activation functions. Deep neural networks do not need to spend extra time calculating errors during the training process because of the constant.

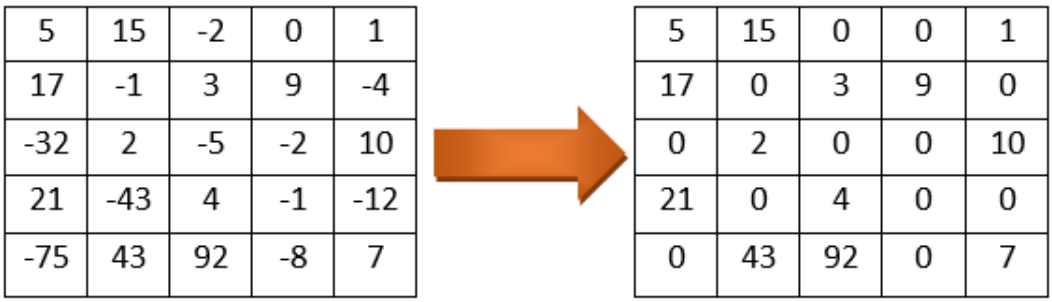


Figure 2.2: Rectified Linear Unit

2.1.3 Dropout

In a neural network, dropout means the removal of units. Dropout is a method of dealing with overfitting. The main concept is dropping a unit randomly during training from the neural network. Dropping a unit refers to withdrawing it from the network for some moment. The rate is passed as a statement to the dropout layer in our model. It reflects the percentage of input units that would be dropped. For example, if we set the rate to 0.5, which ensures that in each epoch, 50% of the neurons in that layer would be lowered at random. If there were 512 units in the dense layer, then only 256 would be trained in the second dense layer after 50% of the neurons are removed. The 256 neurons are chosen at random and dropped. Since the algorithm performed this process at random, certain neurons were dropped rather than others. We repeat this step many times to ensure that each unit follows a remarkably similar procedure.

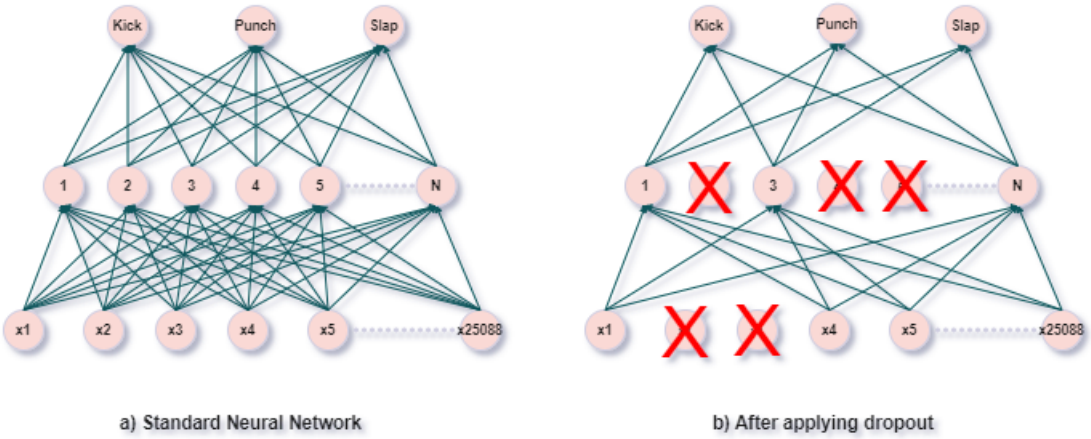


Figure 2.3: Dropout

2.1.4 Softmax

Softmax is an activation function. It is also known as the soft argmax function. It generalizes the logistic function into multiple dimensions. We can use the sigmoid

function to tell which class our output belongs to. Either it belongs to class 0 or it belongs to class 1. However when we have multiple classes or classes greater than 2, softmax function performs better than other activation functions. Softmax function will convert all of the score values into a normalized probability distribution. In a neural network, most of the time softmax is used in the last activation function. This is how the softmax function looks.

$$\sigma\left(\overrightarrow{z}\right)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.2}$$

Here Z is the input vector. K is the number of classes. $e^{z}i$ is the standard exponential function for input vectors. And $e^{z}j$ is the exponential function for output vector.

2.1.5 Fully Connected Layer

We know that an image is composed of smaller details of features. Convolutional neural networks leverages this fact and utilizes its various layers for analyzing each feature in isolation thereby informing a decision about an image as a whole. A fully connected layer in CNN is the one that takes the end result of the convolution or pooling layer while a flattened layer and reaches a classification decision. In a fully connected layer each input is connected to each output by weight. Traditional CNN would be unable to spit out the predicted classes Without a fully Connected layer. So basically after applying all the layers like convolution, RelU, dropout, flattening etc an artificial neural network is then used to process the flattened feature map. So it is like adding an artificial neural network to a convolutional neural network. In the output layer we get the predicted classes.
In Figure 2.4, assumption here is that we have performed all the convolution, ReLU, flattening operations before feeding the flattened feature vector to this ANN which represents a fully connected layer. So this ANN has 25088 input neurons. In the first and second fully connected layer we have N number of neurons. It also has three output neurons which are kick, punch and slap. These three classes can be represented using just two neurons, that is these two neurons can depict number 1 or 0. So number 01 can represent kick, number 10 can represent slap and number 11 can represent punch. If we had 5 classes then then we would have had 5 neurons in the output layer. Each neuron depicts 1 class so if we have n classes then we should have n neurons in the output layer. Our feature vector data is passed through this network. Let's say it predicted the image of a slap with 90 percentage probability however it was a kick. That means it was predicted wrongly hence it is an error. This error is also known as cost function. So the cost function of the prediction is calculated. It is then back propagated through the system to improve the prediction, that is minimizing the value of cost function as well as optimizing the weights. Let's say we have these hypothetical numbers assigned to this second fully connected layer. These numbers are in between 0 & 1 and hence depicts a probability value of a neuron. It is confident of finding a specific image feature. Zero means that the neuron is not that much confident of finding a feature. Every neuron is detecting their specific features. Let's say the third neuron of the second fully connected layer marked with probability 0.9 is identified as a hand on a face. It is pretty confident that it is a feature of a slap and it is passing this feature to all the neurons. Now it's up to these neurons to understand if the feature is meant for them or not.

Lets again assume there are two other neurons on the second fully connected layers which are firing up. Let's say one is detecting the leg of a man and another one is detecting another man being thrown over. These are also passed to all the output neurons. Since these features belong to kick, this output neuron knows that the results should be predicted as a kick and hence it will be fired. After several such multiple iterations which are called epochs in keras framework the output neuron related to kick learns that these features in fully connected layers fire up when these features belong to kick. On the other hand let's say these other neurons get fired up when the features are related to a slap. So, the output layer actually learns which of the final fully connected neurons to listen to for specific features of an image object. Now if I provide a new image of punch to this network, these neurons which can identify features related to punch get fired up to give a collective vote to output node related to punching and hence classifying it as a punch.

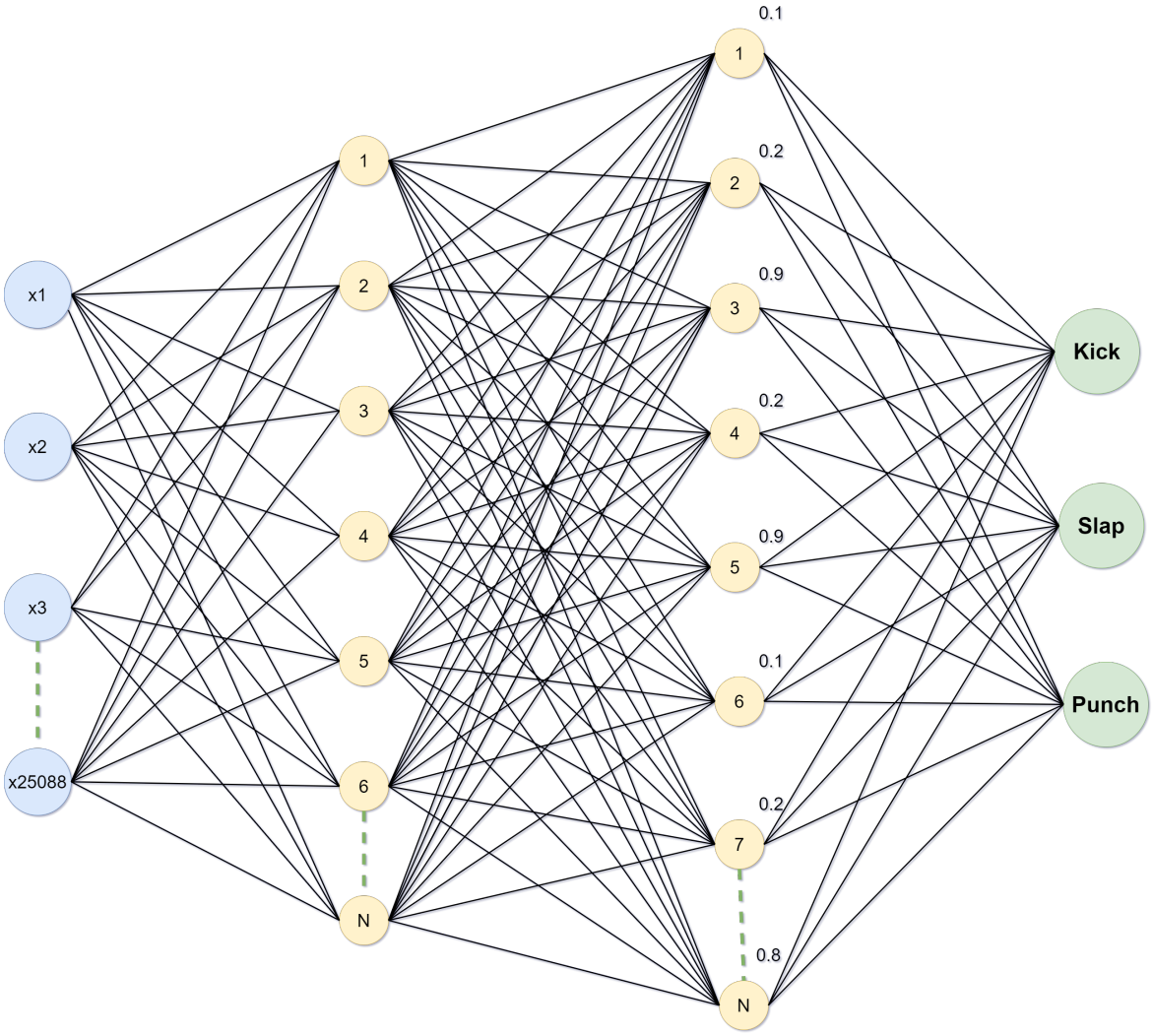


Figure 2.4: Fully Connected Layer

2.2 Related Work

Despite having CCTV in several areas to protect people from any unexpected incidents, unfortunately, it is increasing rapidly because of its lack of effectiveness and they need a significant number of trained supervisors/operators [12]. In this study, the author examined some CCTV footage to determine how to improve the performance of explicit motion information. Here they proposed a pipeline through Two-stream CNN, 3D CNN, and a local interest point descriptor. To start this project, they collected 1000 videos of real-world CCTV-Fights and observed them by their characteristics. Additionally, they used the Visual Feature Extraction as the first stage, which is structured with RGB information. Then the author used a two-stream-based approach (RGB and Optical Flow) performed by 2D-CNN architecture, a 3D-CNN pipeline (with temporal information as the third dimension) using convolutional neural network architecture, and local interest points. They chose to mean average precision (mAP, higher is better) to determine the performance of these given methods. After applying the methods in the CCTV-Fights dataset, the percentage of mAP of Two-Stream is higher than any other method, an indicator of performance improvement. Moreover, they examined CCTV and Non-CCTV video and they observed that the Non-CCTV fights have a higher mAP than CCTV fights. From these experiments, they realized that explicit motion information is more auspicious than the RGB-only methods.

As mentioned earlier, due to the availability of a huge amount of video data through the internet, it is becoming complicated to assess violent activities in real-time. In this paper, the author tried to distinguish videos into two classes violence and non-violence based on audio-visual information [4]. The audio features used 12 different audio features statistics to determine the window length and used Bayesian Network to build binary classification. Using this binary classification using the K-Nearest Neighbor (KNN) algorithm, they determined binary decisions (whether the video is violent or not). Secondly, they extracted Video Features, Motion Features, and Detection Features to get a rich feature set and used them in the KNN Algorithm to determine the probabilities of whether the video is normal activity or higher activity. After the detection performance, they had concluded that the audio-based detection has a higher rate of performance and 17% of the violent videos are not detected properly.

On account of poor-quality video surveillance scenarios, it is very tough to identify whether a video is violent or the movement and behavior are recognizable. In this paper, the author tried to approach a to automatically detect the motion and recognize the movement whether it is violent or not [7]. The author used Optical Flow Images (using a motion-resilient algorithm), Motion Type Detection, and Feature Extraction (using motion magnitude, acceleration, and motion signals) to classify into 8 motion types. After that, they performed a statistical approach on each type and computed them based on motion signals, namely, mean, maximum, minimum, median, and standard deviation. It showed a percentage distribution of different motion types to use in the machine learning algorithm to detect the violation in the video. Finally, over 1000 videos they observe the accuracy of their experimental datasets is 98.5% and their results can naturally detect fighting or violation with

only motion analysis.

Due to the lack of automatic detection of violent scenes, the children can watch violent movies and scenarios on the internet which is a matter of concern for a parent [2]. In this study, the author tried to automatically detect violent videos to prevent the children from watching them. Here, the author selected three stages: segmentation of video into a set of shots, training in a semi-supervised way of low-level visual and auditory features, identifying the high-level audio effects, and determining whether the video is violent using probabilistic output stages. Moreover, the author had used two different algorithms like SCFL and SVM. Here SCFL is a semi-supervised cross-feature learning algorithm utilized in learning the classifier. On the other hand, SVMs are trained to classify audio clips to specify the audio effect each to each SVM model. To evaluate the difference between the SCFL and SVM model the author has experimented with the datasets over several movies and using precision, recall, F1-measure the author identified that the performance of SCFL is better than SVM by a significant margin. Finally, the author claimed that using SCFL the violence of a video can be measured automatically.

In today's generation, the spreading of violent videos throughout the internet is very common which should have to be prevented automatically without human involvement [13]. In this paper, the authors initially observed that CNN is a better approach, which extracts the fine-grained visual details via filters from the static images, which results in an excellent performance in 2D image classifications. The same performance is achieved for videos by using 3 dimensional CNN filters. However, for both cases, the issue arrives as the size of the dataset. Because a substantial amount of training and diverse samples are needed for training deep neural nets. To tackle this problem, the authors proposed a new dataset named Foi-Fight dataset with non-violent content and also gathered the violent video datasets from different sources like VISILAB (600 videos), UCF crimes (128-hour videos), CCTV-Fight, etc. Moreover, he invented a new implementation strategy named zoom to increase the neural network's capacity so that it can speed up the handling of the number of frames and make the classification more careful and precise. To use this strategy, he used 3 inputs i) RGB input for single-stream I3D, ii) RGB and RGB difference input for two-stream I3D and iii) optical flow input for two-stream I3D through LiteFlowNet. After training those datasets with his new strategy he checked the accuracy using T-Accuracy, FN-Accuracy, and FP-Accuracy. Finally, he determined that zoom (his invented strategy) is better than the previous strategies to automatically inspect violent videos from movies or web videos.

In this modern cruel world, an automatic fighting action detector is very important to ensure the security of children and the people [5]. In this paper, the author focused on finding higher accuracy in fight detection. For this purpose, they came up with a new dataset of hockey videos to find the violence in sports footage. The dataset consisted of hockey match scenes with fight and non-fight video data. Here the author used the bag-of-words approach with using the datasets of hockey game videos of National Hockey League, INRIA (contains kicking or punching video), CAVIAR (instance people aggressive behavior videos) to detect aggressive violence. They used two prominent Spatio-temporal descriptors to examine video violence

named STIP and MoSIFT using HOG, HOF, HNF features vectors. After the performance using HOG, HOF, and HNF vectors they found that MoSIFT has higher performance than STIP. Finally, they observed that detecting violence in hockey footage is easier than detecting fights in movies or other actions, and using the bag-of-words approach in MoSIFT can get approximately 90% accuracy.

Chapter 3

Proposed Real Time Action Recognition Method

3.1 System Model

In this project, we have worked with different videos collected from youtube, tv shows and different resources. We have labeled all the videos manually which was a little bit slow process but very efficient in assigning the right labels. After splitting the videos into train, test and validation, we have extracted a few frames from each video. Then we have converted those frames into arrays and normalize the values we get from the images. At the same time we used one-hot encoding on our labels. After finishing this pre-processing part we feed these values into some pre-trained models and get an array of values for each image. After that we again normalized those values to get better solutions. To feed these values into our model we reshape them into 1-dimensional shapes. On our model, we tried to generate outputs same as our one-hot encoded label outputs. The next step was to compare which pre-trained model is generating good answers for our model. In short, we tried to create a solution where we can use existing solutions along with our solutions to get better results. Figure 3.1 represents our proposed workflow.

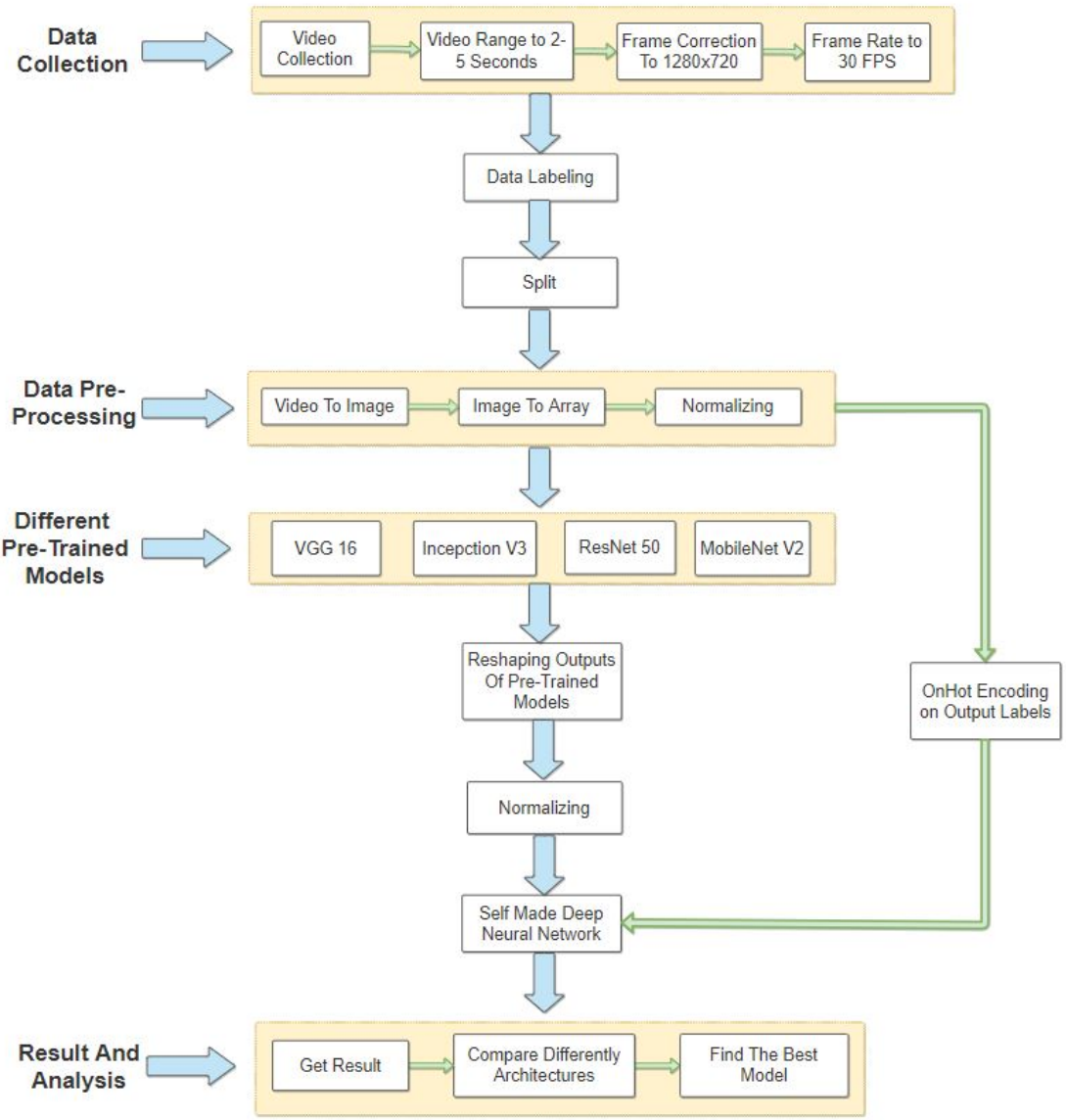


Figure 3.1: System Model

3.2 Data Acquisition and Preparation

We primarily focused on collecting data for three distinct categories, slapping, punching, and kicking for this project. These data were collected mainly from a public video sharing domain, i.e., YouTube, and sampled manually (the frame regions only corresponding to the actions of our interest). Further, we manually labeled the trimmed data into respective class labels. The total number of collected clips is 600 where we have divided clips according to its label. On average, each clip duration ranged from 2 to 7 seconds, which indicates the only period for the actual action. We ensured high-definition resolution which was 1280 x 720 of the videos and a unified frame rate of 30 for all clips.
Figure 3.2 represents intensity of scenes. From Figure 3.2, we can see very little correlation between two images, we need to normalize them to find correlations.

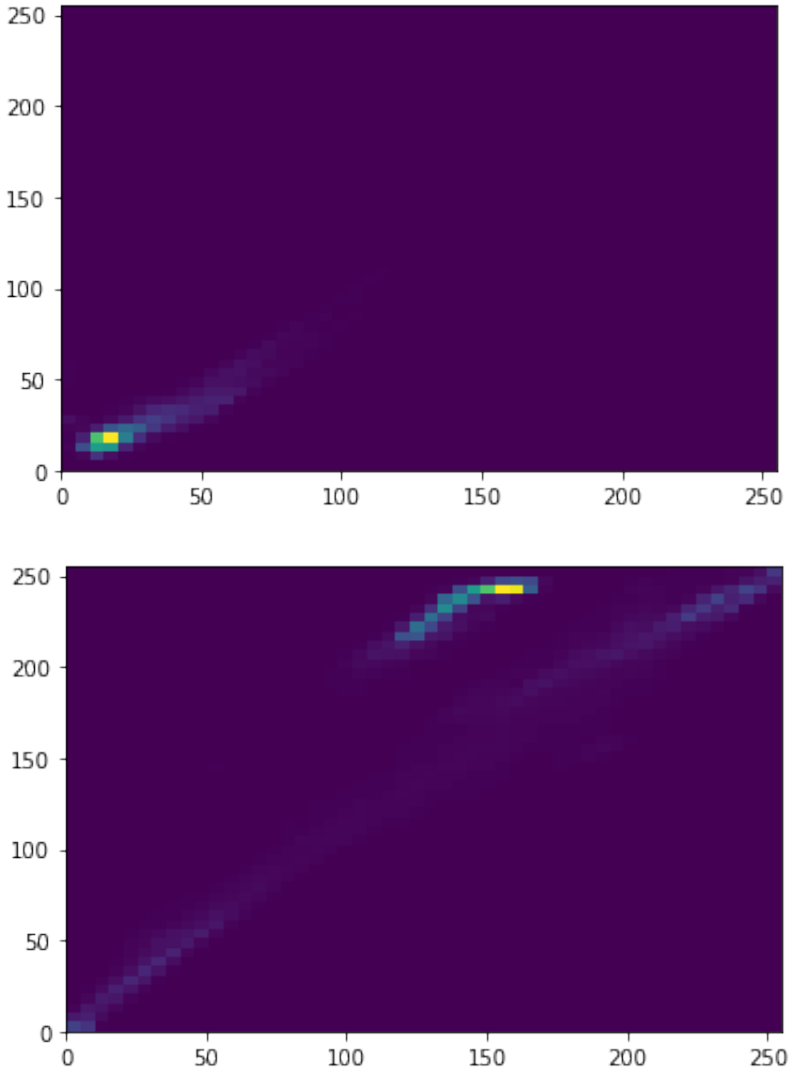
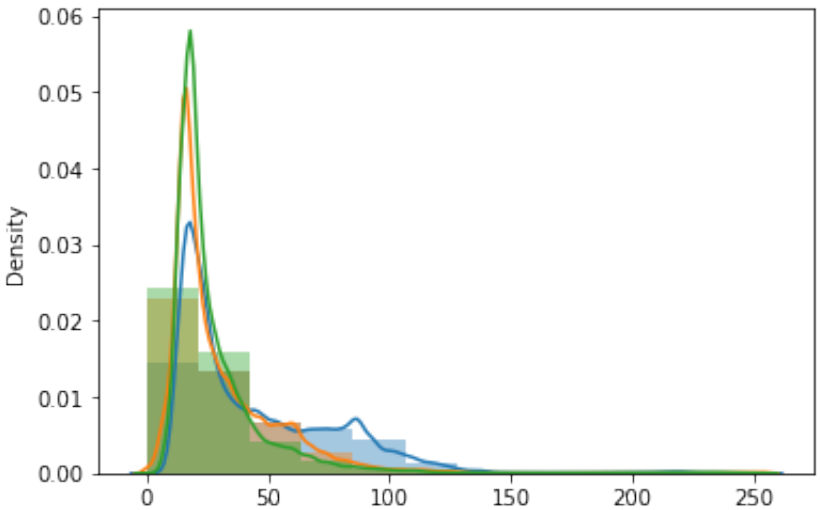


Figure 3.2: Intensity of Scenes



From Figure 3.2, we can see the differences of color value between scenes.

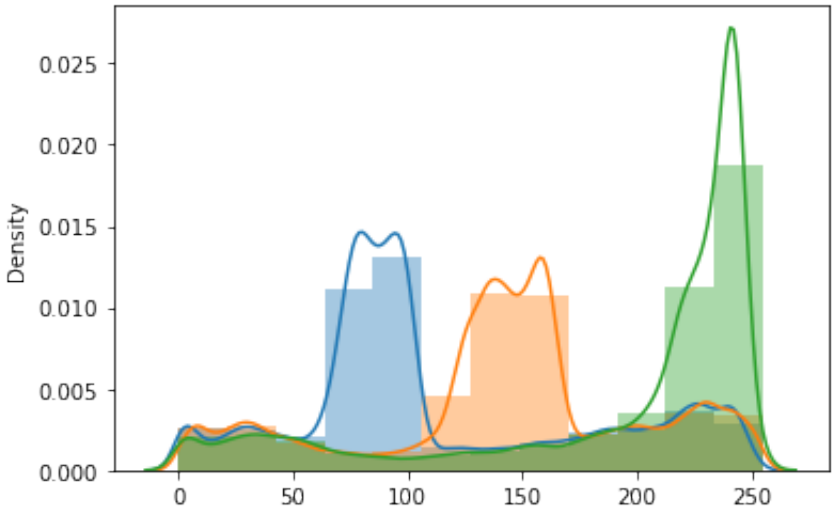


Figure 3.3: Color Value Histogram

3.3 Video To Frame Conversion

In the pre-processing part, we have converted videos to image frames and resize those images using openCV3. Our video to frame conversion algorithm will work as,

Algorithm 1: Video to frame conversion

```
Result: Here will be the result
initialization
if not directory then
    os.makedirs(test.directory)
end
for data in test.dataset : do
    video = videoCapture(data['video'])
    action = data ['action']
    video_name = data ['video']
    frame_rate= video.getframeRate()
    count = 0
    while video is open do
        frame = video.read()
        frame id = video.getframeid()
        if frame_id % frame_rate = 0 : then
            file_name = test_directory + video_name + frame_id+ count +
              action+ 'jpg
            cv2.imwrite (filename, frame)
            count + = 1
        end
    end
end
```

3.3.1 Video to Image Conversion OpenCV

Video is a collection of images. By the term video we understand recording, reproducing, or broadcasting of moving visual images. Like images, videos also have width, height and depth. As video is a representation of visual images, we need to decide at which rate images will be shown in a video. We mention its rate as frame rate per second, in short fps. Fps refers to how many images will be shown in a video per second.

Firstly, we have calculated the frame rate for each video. Working with all frames of a video might not be a good solution for our problem as we are thinking of predicting a real-time situation where working with each frame might lead to a wrong assumption and this process is more time-consuming. So, we have divided the frame rate with each frame number and taken only those frames which set the reminder value to zero. Entire process has been done using openCV3.

Functions and parameters used for Conversion:

- cv2.VideoCapture : Create a video capture object from a given url or Camera.
- cv2.CV_CAP_PROP_FPS : Catches frame rate from captured video. It is a property of get function.
- cv2.CV_CAP_PROP_POS_MSEC : Gives the value of the current position of the video file in milliseconds or video capture timestamp. We need to use this property in the get function to get the frame rate. It is also another property of the get function.
- cv2.VideoCapture.read : Returns 2 values, frame and read status. If read correctly then true else false.
- cv2.imwrite : Takes two parameters. One is the file name, the other one is the frame to be saved.
- capture.release : Releases the captured video object.

Figure 3.5 ,3.6 ,3.4 shows the sample frames that are extracted from the videos.

3.3.2 Image Resizing conversion to array using Keras

Initially, our frame size was (1280×720). After doing some research we found that most of the pre-trained model takes (224,224,3) shape as input. While loading frames, we converted our image into this shape. We used keras preprocessing tools to resize images.
Functions and parameters used for Conversion

- keras.preprocessing.image.load_img : Loads image from a given url. Takes path, target_size, grayscale, color_mode, interpolation as input. Url can't be empty. We give (224,224,3) as target_size.

Figure 3.4: Frames Extracted from videos : Slap

Figure 3.5: Frames Extracted from videos : Kick

- Keras.preprocessing.image.img_to_array : Converts given objects to an array of pixel values.

After converting to an array we normalized all the image arrays to extract more information from it.
Figure 3.7 & 3.8 shows the state of before and after normalizing the images. Figure 3.7 shows contrast of the images are not covered the whole area of the Histogram properly. From 3.7, we can understand that interesting part of the contrast are in 0 and 1. So, to extract more information from images we need to reduce the contrast.

Figure 3.6: Frames Extracted from videos : Punch

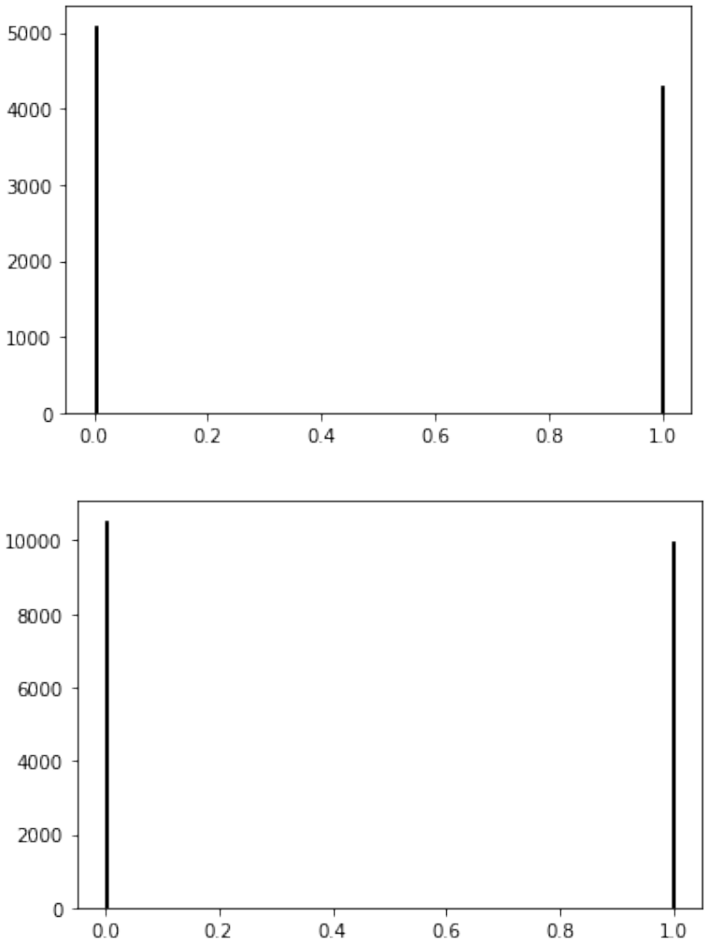


Figure 3.7: Before Normalization

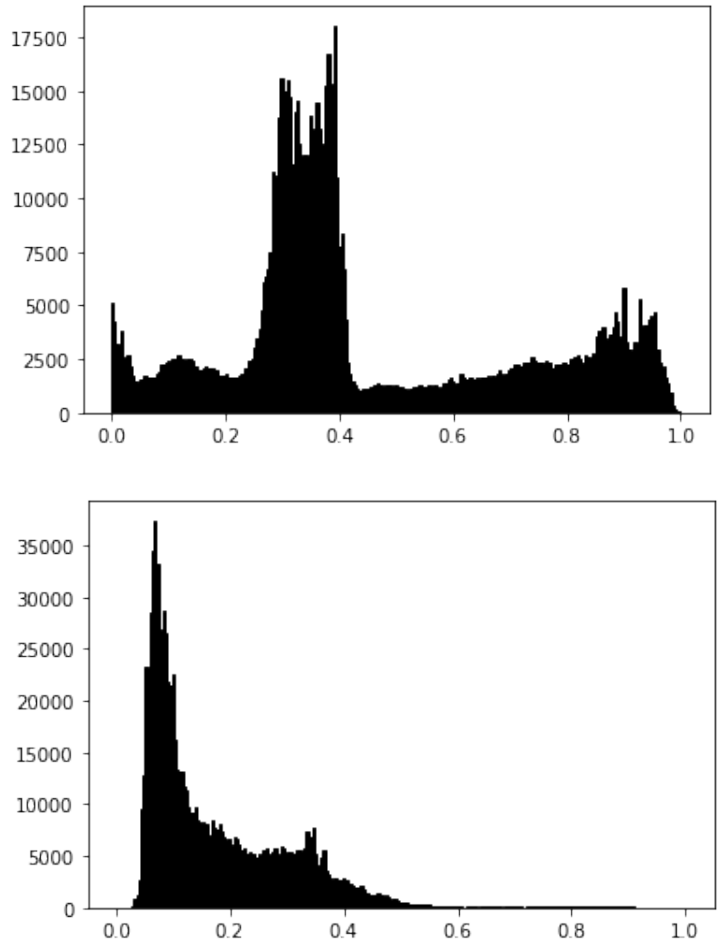


Figure 3.8: After Normalization

After normalizing images we can clearly find the regions to differentiate between images in 3.8.

3.4 Data Classification

As previously mentioned, in our work we have detected 3 types of actions. Which are kick, slap and punch. At first, we manually assigned labels for each video data. While converting video to image we stored the label with the frame by including action to the frame name. We have used one hot encoding to create output classifications. We have created a separate output row for each label and populated it with binary 0 and 1 according to the previously created labels.

3.5 Model Specification

3.5.1 VGG

Over the past few long years, Spiking Neural Systems have ended up well known as a conceivable pathway to empower low-power event-driven neuromorphic equipment. In this paper, they propose a novel algorithmic method for creating an SNN with

profound design and illustrate its adequacy on complex visual acknowledgment issues such as CIFAR-10 and ImageNet. It has 14 million hand-annotated pictures of what is within the picture. Let's illustrate the design of VGG. VGG stands for the Visual geometry group. Subtracting the average RGB esteem calculated within the preparing set from each pixel is as it were prepared. Our input ConvNets size is fixed, which is 224 × 224 RGB picture. The picture is tested by a number of convolutional Layers, utilizing a really little open field: 3 × 3 (which is the smallest measure to capture the idea of that 360 degree area) and utilizing 1 × 1 convolution channels also. It can be seen as a straight change of the input channels. The convolution walk is 1. The spatial determination is protected after convolution, i.e. the cushioning is 1 pixel for 3 × 3 conv. Max-pooling is performed over a 2 × 2 pixel window, with walk 2 [6].

VGG-16

The (VGG) Visual geometry group may be a well known DCNN show, which was excerpted by K.Simonyan and A. Zisserman in 2014. VGG formed 92.7% as the top 5 test precision at OLSVRC competition. The most key part is expanding the profundity of the organize with exceptionally small(3 × 3) convolution channels by two convolutional layers are utilized ceaselessly with a corrected straight unit (ReLU) as actuation work taken after by a max-pooling layer, a number of completely associated layers with ReLU and soft-max as the ultimate layer. VGG Net has three categories depending on the total number of layers existing within the engineering, they are VGG-11, VGG16 and VGG-19. The essential VGG-16 structures is shown in Fig3.9.

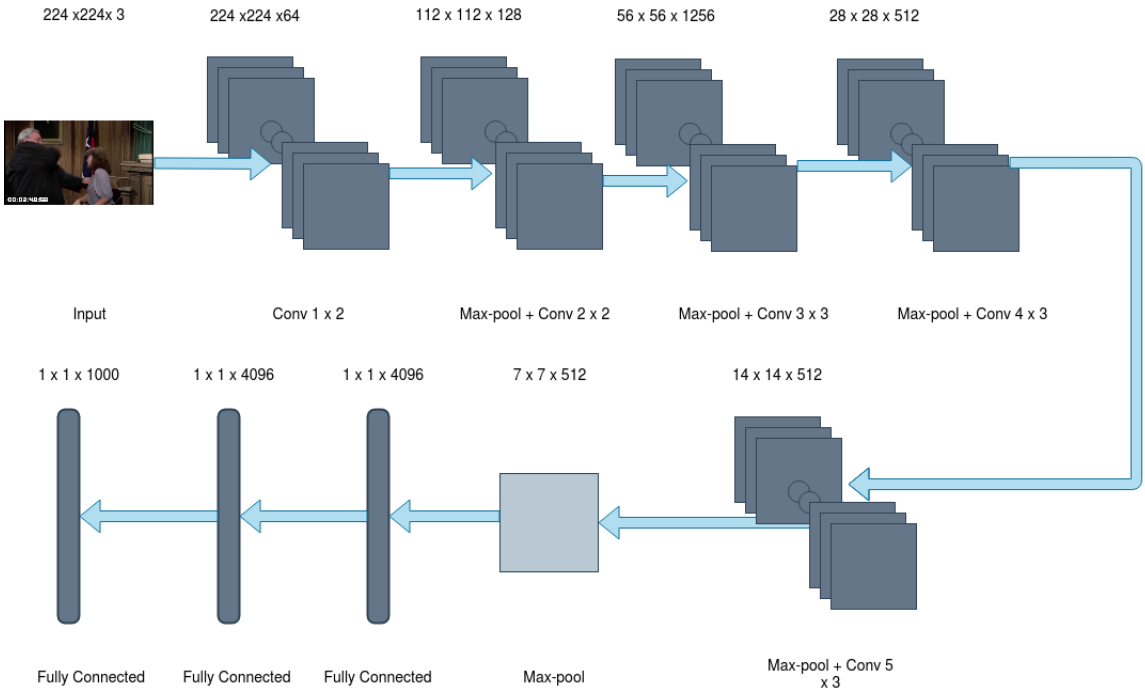


Figure 3.9: VGG16 Basic Architecture

VGG-16 and VGG-19 can be 16 and 19 layer projects respectively. Meaning The VGG-16 design consists of 16 convolutional layers and the VGG-19 consists of 19

convolutional layers. Compared to the "VGG-19" organization program, the "VGG-16" organization program has less weight. The "VGG-16" metrics and partial counts are closely related to classifier and discard layer regularization.

3.5.2 Inception

The Inception model was first introduced in 2014 and was used in the GoogLeNet model. It is a pre-trained convolutional neural network. The inception model is a very simple yet powerful architectural unit. It allows the model to learn both parallel filters of differing sizes and parallel filters of the same size. In simple words it allows learning at multiple scales. The most trademark of this design is the improved utilization of the computing resources inside the network which was accomplished by a carefully crafted design that permits for expanding the depth and width of the network whereas keeping the computational budget consistent [1].
The key idea behind this model is the inception block. The purpose of this model is to act like a multi-level feature extractor. It means it will be able to compute 5×5, 3×3 and 1×1 convolutions within the same module. Before being fed into the next layer the outputs of these filters are stacked along the channel dimension. It was also known as GoogLeNet. 3.10 & 3.11 Figures represent the general architecture of the inception model.

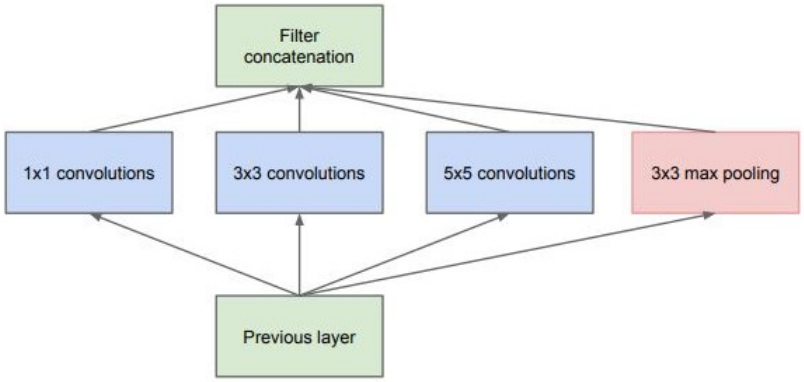


Figure 3.10: Inception module : Naive Version [1]

InceptionV3

Co-authored by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jonathon Shlens Inception V3 was proposed in 2015 through their paper "Rethinking the Inception Architecture for Computer Vision". By modifying the previous Inception architectures the authors focused on burning less computational power. It was trained using a dataset of 1,000 classes which are from the original ImageNet dataset. Inception v3's architecture is progressively built, step-by-step. At first factorized convolutional which helps to reduce the computational efficiency. Then smaller convolutions which replace bigger convolutions with smaller convolutions. Later asymmetric convolution where a 3×3 convolution could be replaced by a 1×3 convolutional followed by a 3×1 convolution. Then an auxiliary classifier which is a small CNN inserted between layers during training. And finally grid size

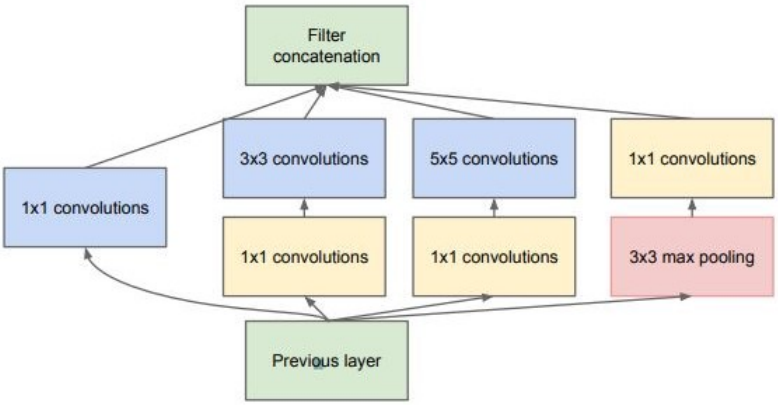


Figure 3.11: Inception module: Dimension Reductions [1]

reduction which is usually done by pooling operations [9]. Our input image shape was 224×224×3 and output shapre from the inception model was 5×5×2048.
3.12 Figure represent the general architecture of of Real Time Action Recognition on inception model.

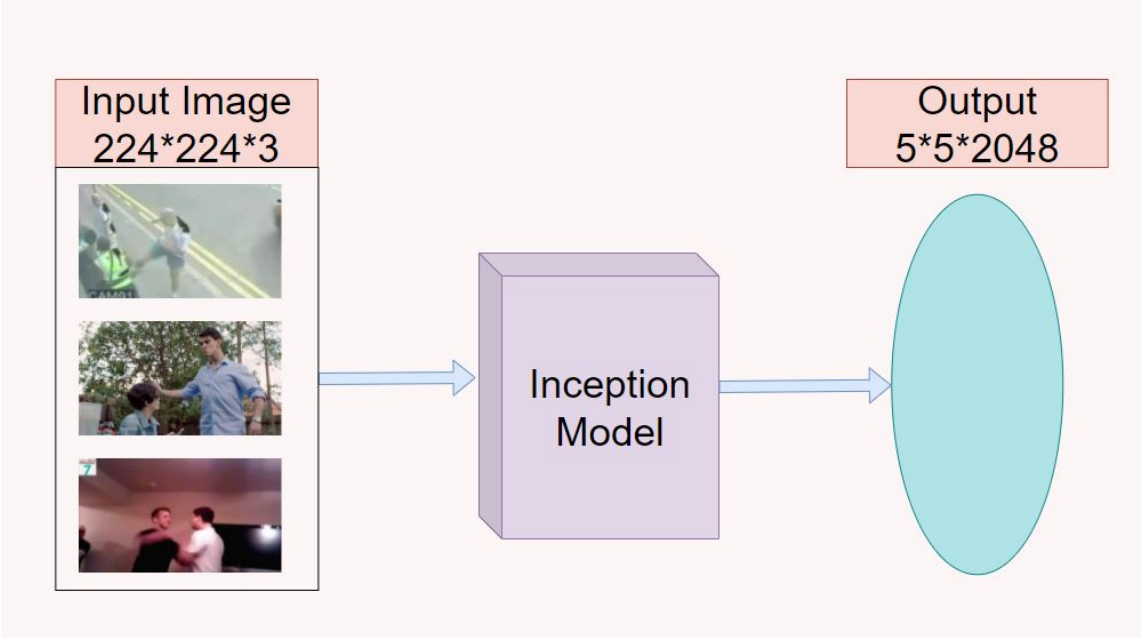


Figure 3.12: Inception V-3 General Architecture of Real Time Action Recognition

Resnet, short for Residual Networks, is a classic neural network ResNet50 is a variant of the Resnet Model. Resnet is one of the most popular neural network designs that have ever been published with over 20,000 citations. Resnet is used as a backbone for many computer vision tasks. ResNet50 won the ImageNet challenge in 2015.

3.5.3 ResNet50

Deep Convolutional neural networks are generally good at identifying features from images. Also stacking more layers provides a higher accuracy. So the idea is that shouldn't building better neural networks as easy as adding more layers to the network. However, the authors of resnet50 state that if you just continue to concatenate convolutional layers on top of activations and batch normalization the training will eventually get worse not better [8]. To address this problem the authors came up with a deep residual learning framework. Resnets are built out of residual blocks. If you consider a shallow architecture and it's deeper counterpart with more layers, theoretically all the deeper models would need to just copy the output from the shower model with identity mappings. So the construction solution suggest that a deeper model should produce no higher error than the shallow counterpart. However, the identity functions aren't an easy function to learn. So therefore, the residual functions formulate the layers by having a reference to the input through these identity or skip connections. It means, if it needed to push the layer down to zero it could easily do it in this framework. One of the interesting thing with resNets is if the previous layer dimensions don't match the input to the next layer they propose different schemes for up sampling the previous input layers through identity skip connection. 3.13 Figure shows how the authors compared ResNet50

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.13: Resnet50 compared with other ResNet Models [8]

with other ResNet models. In the resnet experiments the authors tested 152 layers on image net and this gets state-of-the-art results. This is eight times deeper than VGG nets but in terms of the floating-point operation measurement it actually has less computation. It uses a batch formalization after each convolution and before activations. Also it uses the batch size of 256. However it does not use dropouts.
Figure 3.14 shows ResNet50 architecture on Real Time Action Recognition where our input image shape was 224×224×3 and output shape was 7×7×2048.

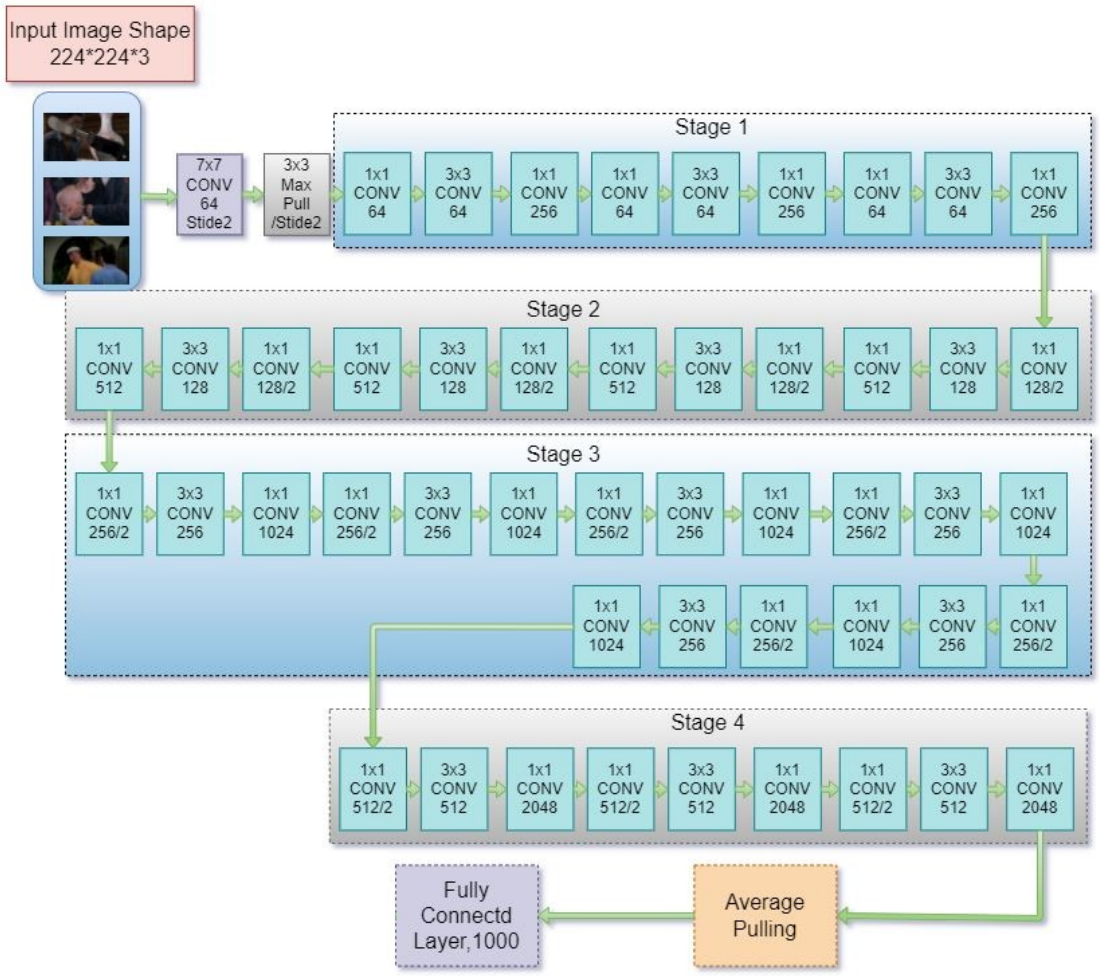


Figure 3.14: ResNet50 Architecture of Real Time Action Recognition

3.5.4 MobileNet

MobileNet was first introduced in 2017 by google researchers for mobile and embedded vision applications. According to the researchers, it is faster in performance than many other popular models despite being vastly smaller in size.

MobileNet is a class of lightweight deep convolutional neural networks [10] . It uses depth wise separable convolutions. It is about 10x faster than VGG-16 and 3x faster than the Inception image classification pre-trained model. Despite being fast it is very small in size. For example, the VGG-16 network on disk takes about 553 megabytes. However, the largest MobileNet is about 17 megabytes. Because of its size MobileNet is considered as a great deep learning model that can be used on mobile devices. In MobileNet at first the convolution layer is split into two parts. In the first part the depthwise convolution layer filters the input and then the 1×1 (or pointwise) convolution layer filters those values to create new features. Together they form a depthwise separable block. It works the same as traditional convolution however it is much faster. In the very first layer MobileNet contains a regular 3×3 convolution. There are no pooling layers in between the depthwise separable blocks. MobileNet uses Relu6 as its activision function as ReLu6 is more robust than regular ReLu. Also the convolution layers are followed by batch normalization. Figure 3.15 represents the general architecture of MobileNetV-1.[10]

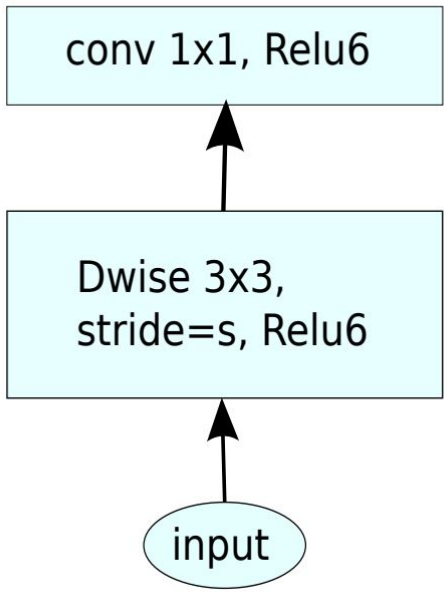


Figure 3.15: MobileNet V1 General Architecture [11]

MobileNetV2

MobileNetV2 was introduced in 2019 by google researchers which was an improved version of MobileNet. It still uses depth wise separable convolutions. However the

authors brought changes to its main building block. The authors added expand layer, projection layers and residual connections to its architecture. This time the authors used three convolutional layers in the block. The last two are depthwise convolution that filters the inputs and the 1×1 pointwise convolution layer. However, this 1×1 layer now has a different job. It makes the number of channels smaller. This layer is now known as the projection layer. The first layer is the new block. It is also a 1×1 convolution. It expands the number of channels in the data before it goes into the next layer. It is known as the expansion layer. It is the opposite of the projection layer. Figure 3.16 represents the general architecture of MobileNetV-2. [11]

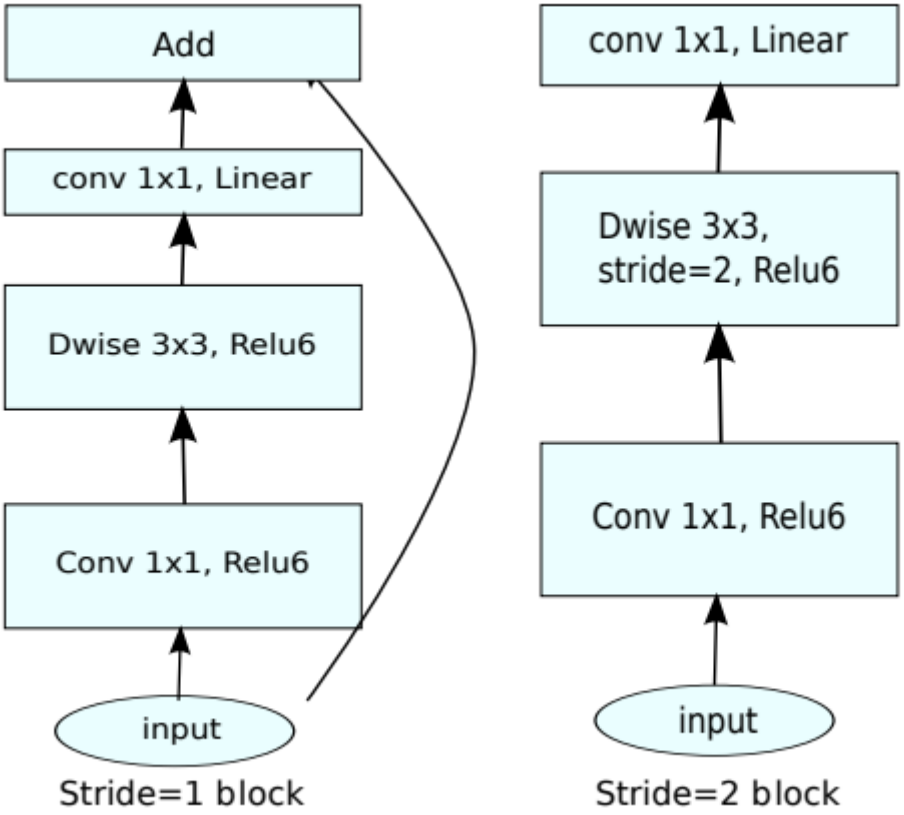


Figure 3.16: MobileNet V2 General Architecture [11]

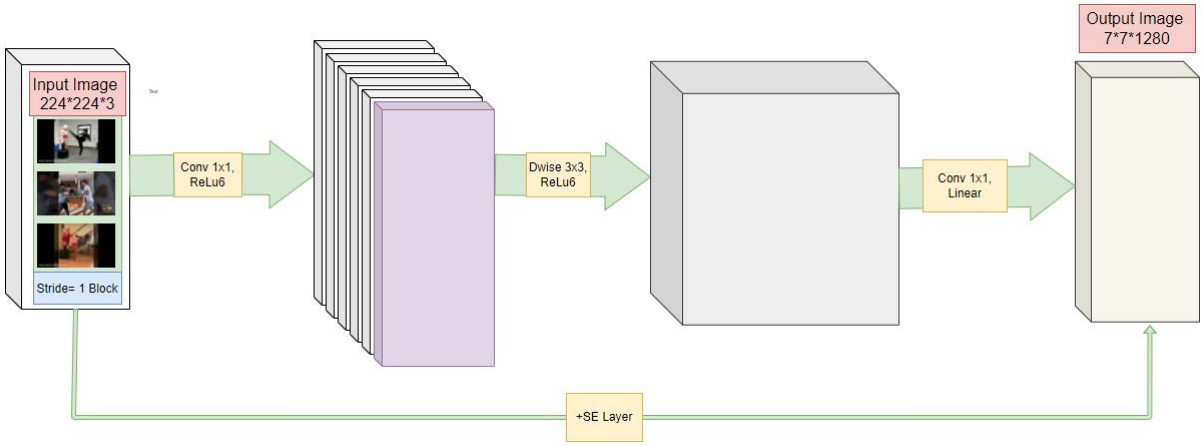


Figure 3.17: MobileNet-V2 Architecture of Real Time Action Recognition

Figure 3.17 shows MobileNetV-2 architecture on Real Time Action Recognition where our input image shape was 224x224x3 and output shape was 7×7×1280. Here first we perform 1×1 convolutions on our input data and in return we receive a much higher dimensional feature set and then we do 3x3 depthwise convolutions on each of it individually to get the same dimension. Finally we perform 1×1 convolutions to string back to original size and add them together.

Chapter 4

Implementation and Result

4.1 Applying proposed Architectures

In our architecture, we feed the pre-processed data into pre-trained architecture. In this part our input dataset shape was the same for all pre-trained models we have worked with. We have got an output containing values which needed to be reshaped. So we normalized them to feed into our model. We calculated accuracy, Confusion matrix precision, recall and F1 score from the final output we have got from our model for each pre-trained model.

4.2 Self designed DNN architecture

Input Layer :
Our input layer of our model works only on image date. In this layer our model takes the output from the image data which is converted from the video dataset that we collected. To work in this layer we convert a three dimensional matrix in a single dimension. For example, if the output of pre-trained models is (8, 8, 64) than the input layer will takes (8 x 8 x 64) = 4096.

Dense Layer :
Several dense layers are used in our model. Another name of dense layer is fully connected layer . In figure 4.1, we apply 5 dense layers in our model, among them 4 with ReLU activation and 1 with softmax activation.

Dropout Layer :
Dropout layer is another important layer for our model. We add a dropout layer to save the model from overfitting. We set the dropout rate 0.5 in every dropout layer that means we drop 50% of input units randomly. In figure: 4.1, in between dense layer 5 and danse layer 6 we add a dropout layer at a rate of 0.5. Which drops the input unit from 1024 to 512. We add a dropout layer in between every two dense layers.

```
]: # The input shape will be 25,088
model = Sequential()
model.add(Dense(1024, activation='relu', input_shape=(25088,)))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 1024)              25691136
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 387
```

Figure 4.1: Self designed DNN model

Activation Function :
In our self design DNN model we use two activation functions. One is ReLU and another one is softmax.
ReLU :
We use the ReLU activation function in our model to remove the negative input value. The mathematical expression of ReLU is :
ReLU = max(0,x) This function of ReLU is very useful for our model. We used ReLU activation function in most of the dense layers of our self design model. This activation function fits perfectly for our model.

Softmax :
In the last danse layer of our model, we use the softmax activation function. We apply the softmax activation function in our model to convert real value into probability. The Softmax activation function assists us in obtaining more accurate results.

Our test prediction algorithms will work as,

Algorithm 2: Test prediction algorithms

```
Result: Here will be the result
initialization
os.makedirs(test.directory)
for data in test.dataset : do
    video = data ['Video_url']
    action = data ['action']
    video_name = data ['Video_url']
    freme_rate = video.getframeRate()
    count = 0
     while video is open do
        frame_id = video.getframeid()
        frame = video.read()
        if frame_id % frame_rate = 0 : then
            file_name = test_directory + video_name + frame_id+ count +
              action+ ".jpg"
            cv2.imwrite (filename, frame)
            count + = 1
        end
    end
    images = test_directory.images
    for image in images : do
        prediction_list = np.array(image)
        prediction_list = base_model.predict(prediction_list)
        prediction_list.reshape()
        prediction.append(model.predict (prediction_list)
    end
    most_predicted = prediction.value_count.max_value()
    predicted.append(y_train.columns.values [most predicted])
    test_directory.clean()
    prediction.clean()
end
```

4.3 VGG-16 Implementation and Result

4.3.1 VGG-16 Accuracy and Loss:

Using VGG16 as pre-trained model with 50 epochs and batch size 128 we achieved an accuracy of 92.68%

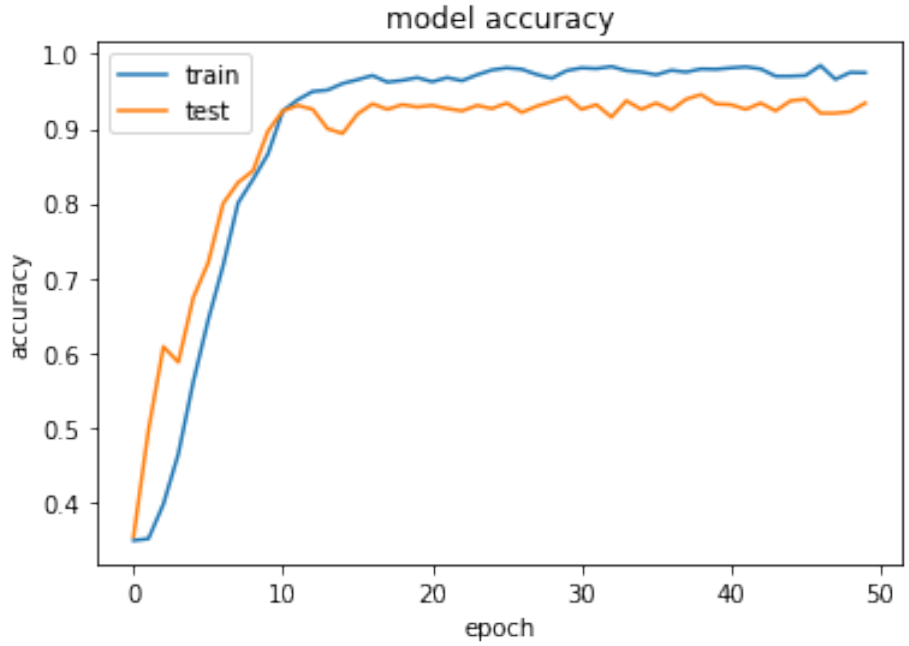


Figure 4.2: VGG16 Accuracy

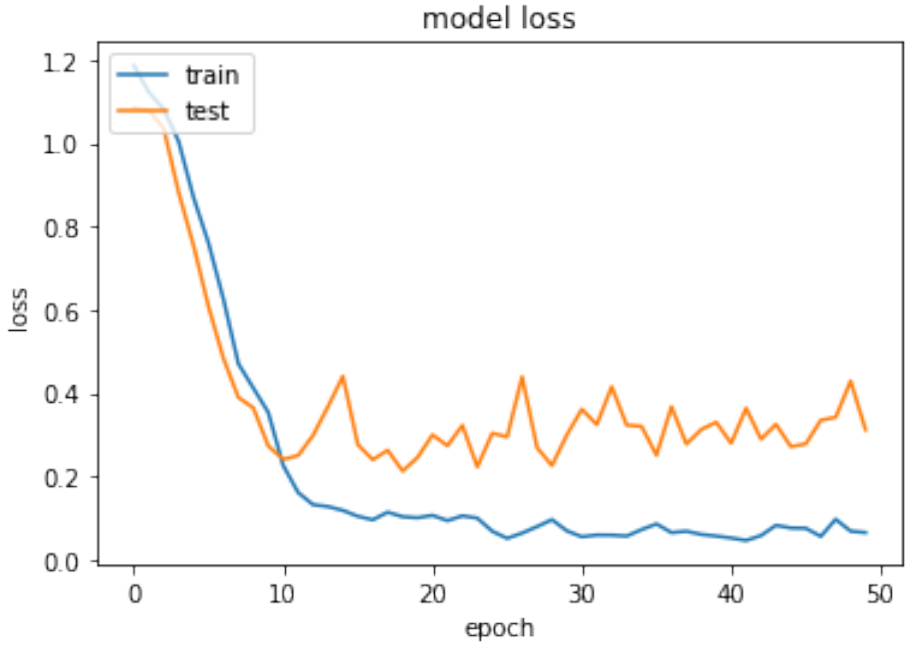


Figure 4.3: VGG16 Loss

From 4.2 & 4.3 graphs, we can see that, model's accuracy has become more linear after 90 which indicates accuracy is between 90to 95 and the trains' loss graph has become more linear after 10 to 20 epoch where test loss fluctuate in certain regions from that we can come to a conclusion that we have come to a conclusion that it has also reached a point from where it will not change drastically.

4.3.2 Precision, Recall & F1 Score Interpretation:

Precision:

Kick	**Punch**	**Slap**
0.92	0.88	0.97
VGG16 Precision Accuracy: 0.92		

Table 4.1: VGG16 Precision

Values from the precision table represent the good performance of our model.

F1 Score:

Kick	**Punch**	**Slap**
0.94	0.91	0.93
VGG16 F1 Score Accuracy: 0.92		

Table 4.2: VGG16 F1 Score

The F1 score table indicates good performance of our model.

Recall:

Kick	**Punch**	**Slap**
0.95	0.93	0.90
VGG16 Recall Accuracy: 0.92		

Table 4.3: VGG16 Recall

Recall table indicates good performance of our model.

4.3.3 Confusion Matrix:

From the 3 confusion matrix, we can see the number of false-positives and the number of false-negatives is very little compared to the correct output.

Confusion Matrix - Kick

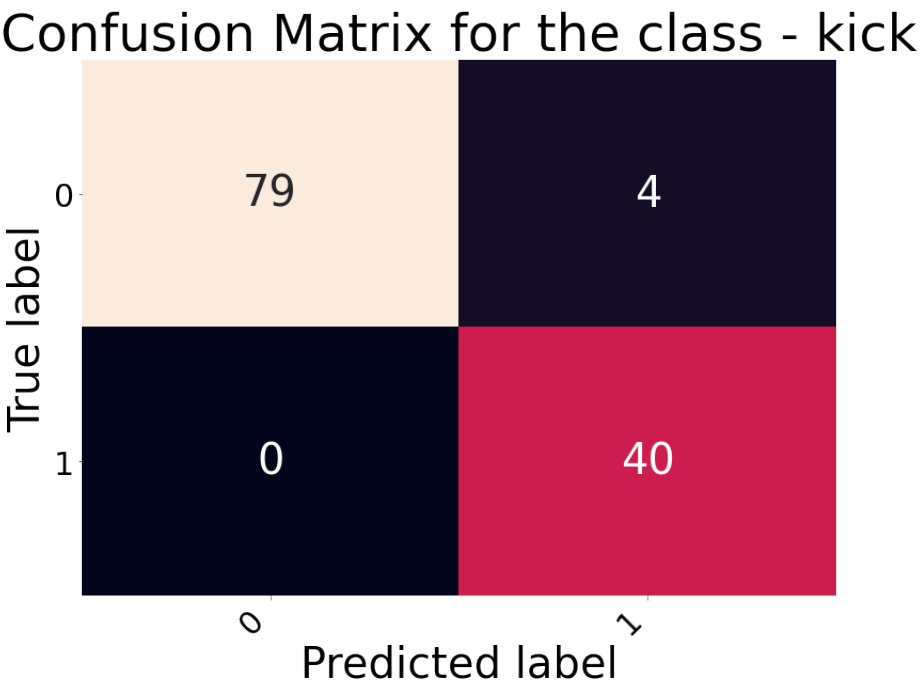


Figure 4.4: VGG16 Confusion Matrix - Kick

Confusion Matrix - Punch

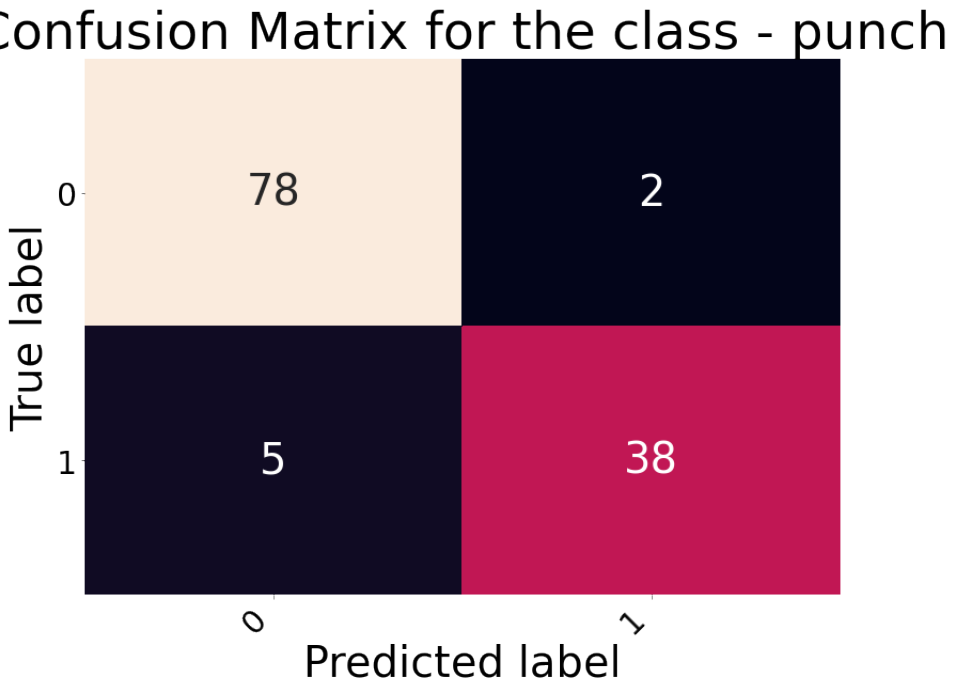


Figure 4.5: VGG16 Confusion Matrix - Punch

Confusion Matrix - Slap

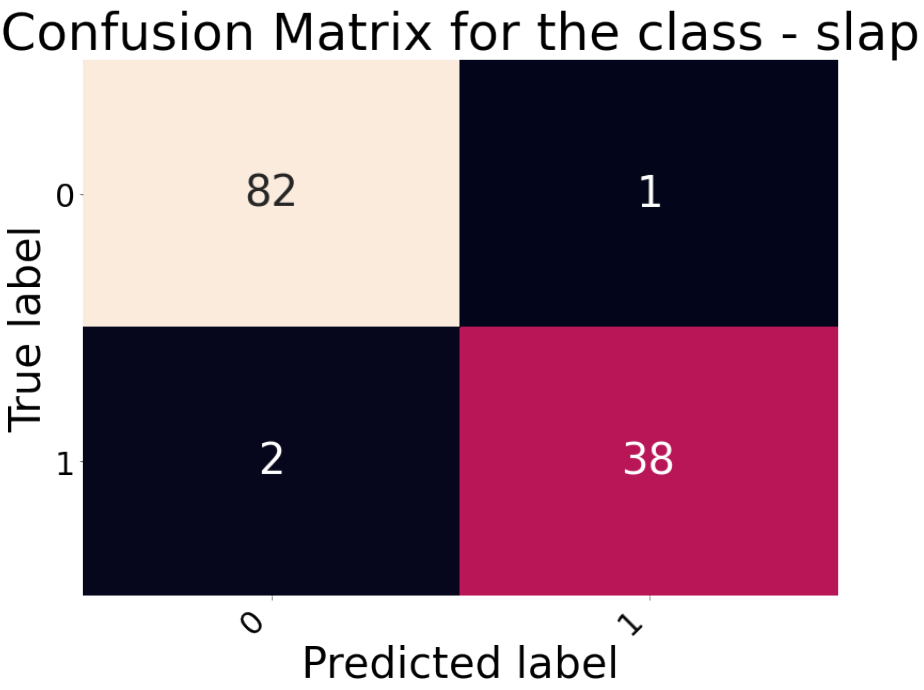


Figure 4.6: VGG16 Confusion Matrix - Slap

4.4 Inception-V3 Implementation and Result

4.4.1 Inception Accuracy and Loss:

Using Inception-V3 as pre-trained model with 50 epochs and batch size 128 we achieved an accuracy of 88.61%

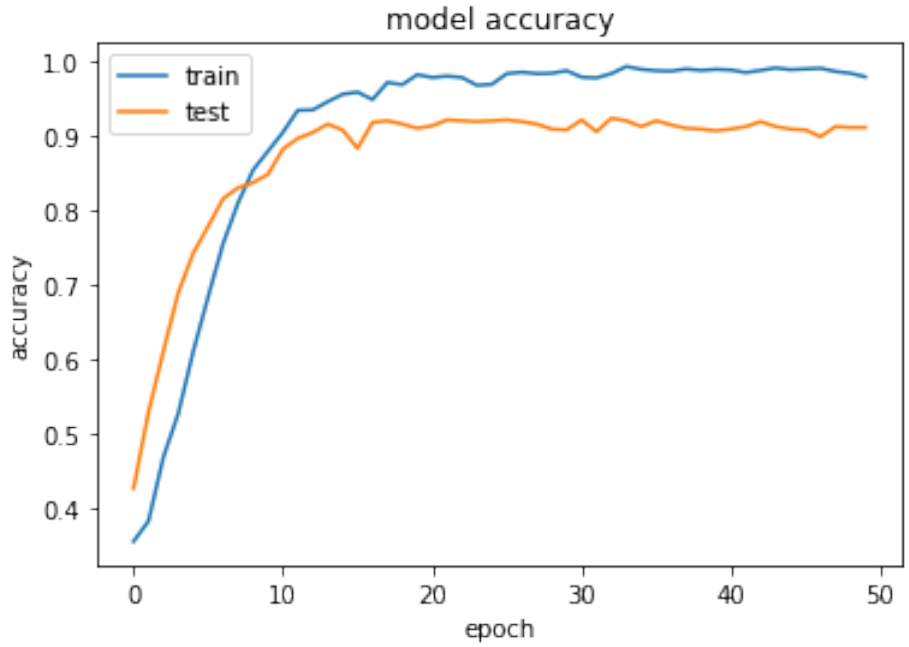


Figure 4.7: Inception-V3 Accuracy

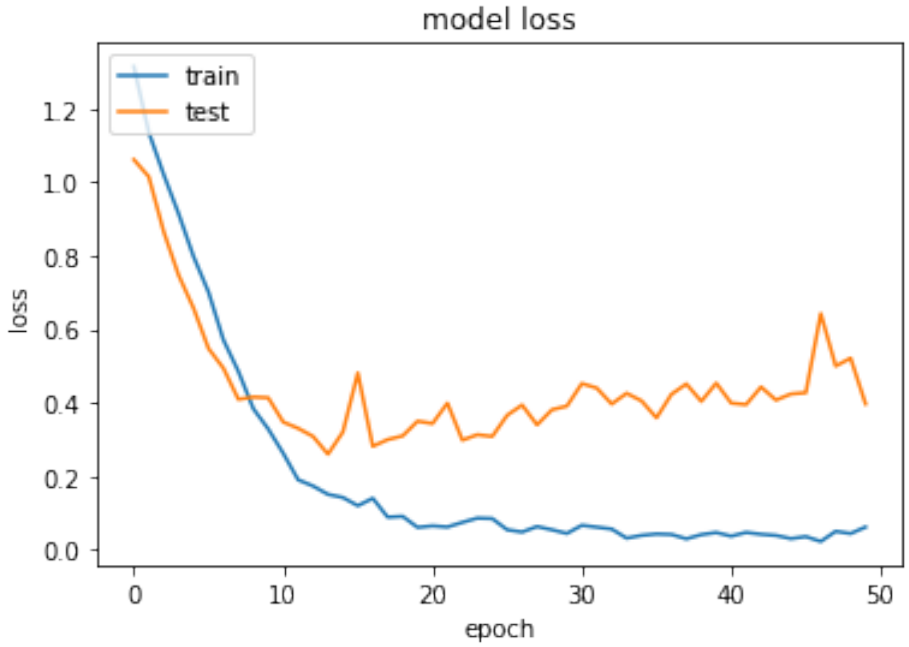


Figure 4.8: Inception-V3 Loss

From 4.7 & 4.8 graphs, we can see that model's accuracy has become more linear around 90 which indicates accuracy is between 88 to 92 and the train's loss graph has

become more linear after 10 to 20 epoch where tests' loss fluctuation has increased after 40 epoch which makes this structure is not effective as VGG16 for our model.

4.4.2 Precision, Recall & F1 Score Interpretation:

Precision:

Kick	**Punch**	**Slap**
0.87	0.82	0.97
Inception-V3 Precision Accuracy: 0.88		

Table 4.4: Inception-V3 Precision

Values from the precision table represent the good performance of our model.

F1 Score:

Kick	**Punch**	**Slap**
0.86	0.85	0.94
Inception-V3 F1 Score Accuracy: 0.88		

Table 4.5: Inception-V3 F1 Score

The F1 score table indicates good performance of our model.

Recall:

Kick	**Punch**	**Slap**
0.85	0.88	0.92
Inception-V3 Recall Accuracy: 0.88		

Table 4.6: Inception-V3 Recall

Recall table also indicates good performance of our model.

4.4.3 Confusion Matrix:

Confusion Matrix - Kick

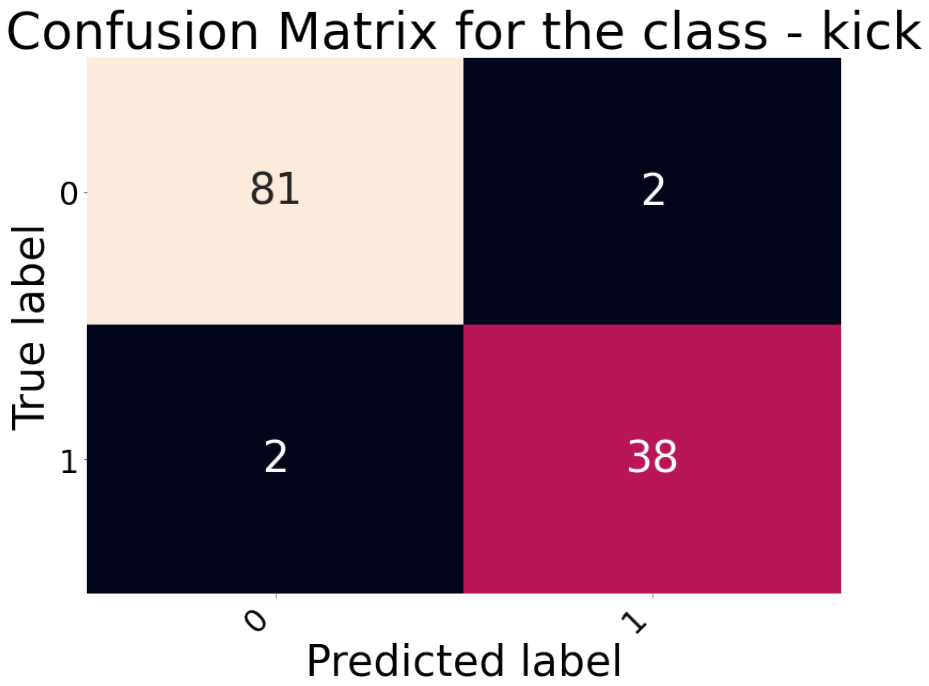


Figure 4.9: Inception-V3 Confusion Matrix - Kick

Confusion Matrix - Punch

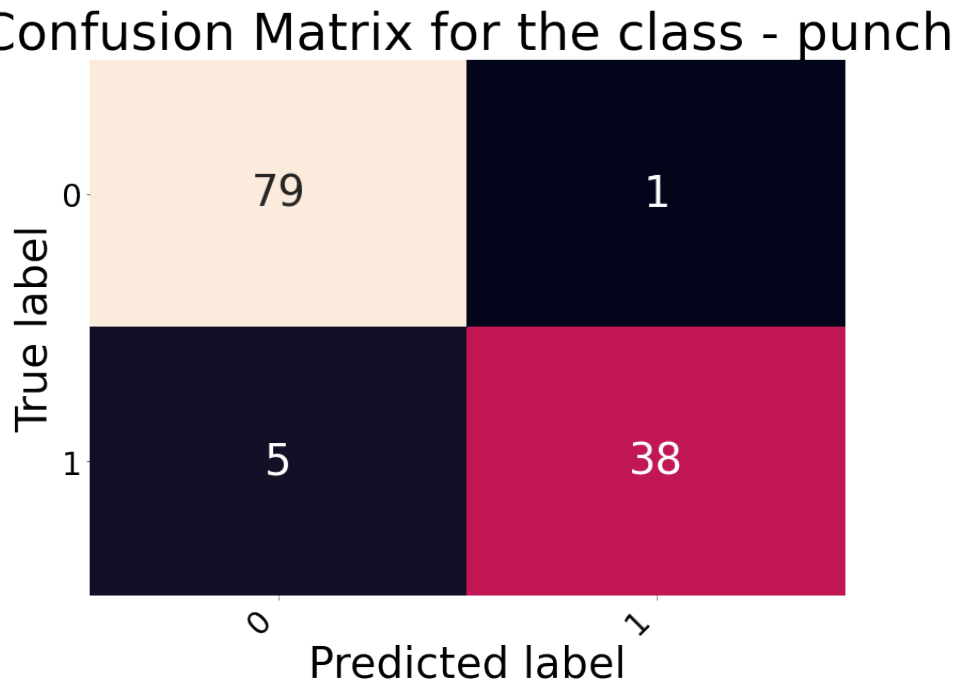


Figure 4.10: Inception-V3 Confusion Matrix - Punch

From the 3 confusion matrix, we can see the number of false-positives and the number of false-negatives is very little compared to the correct output. Though this pre-trained model is not as good as VGG16 for our model.

Confusion Matrix - Slap

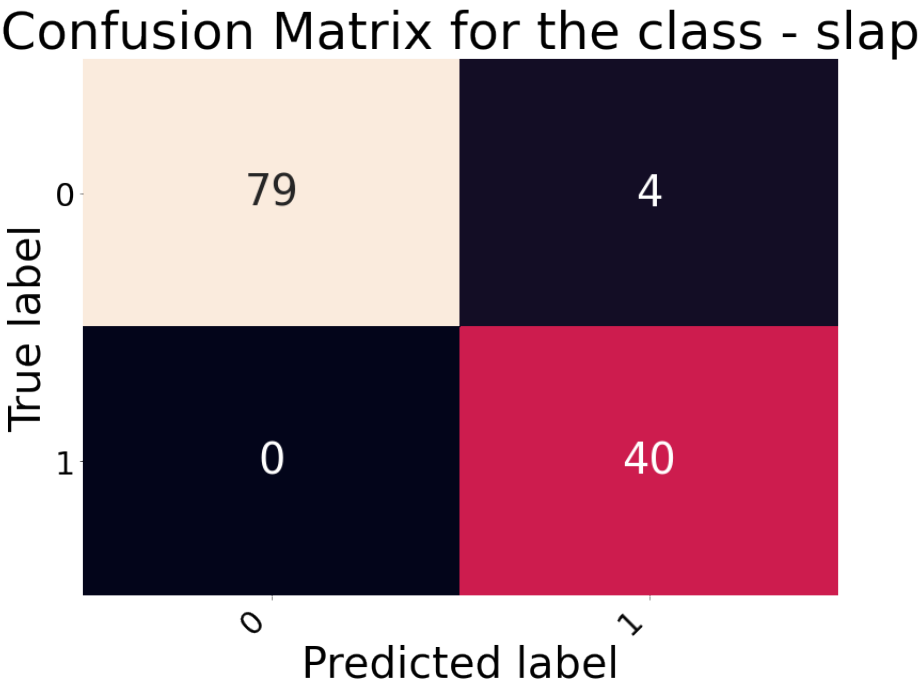


Figure 4.11: Inception-V3 Confusion Matrix - Slap

4.5 ResNet50 Implementation and Result

4.5.1 ResNet50 Accuracy and Loss:

Using resnet50 as pre-trained model with 50 epochs and batch size 128 we achieved an accuracy of 68.29%.

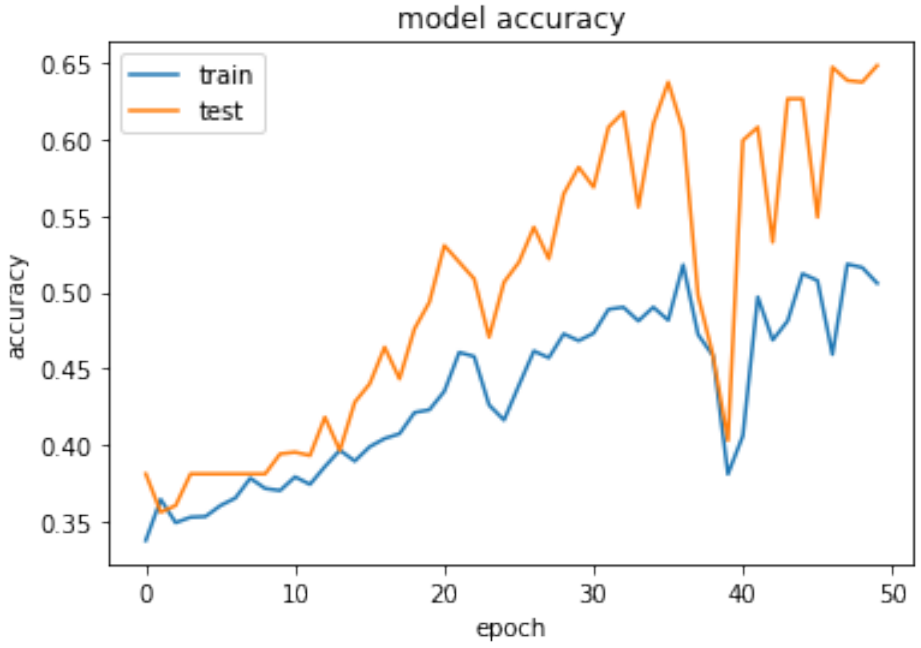


Figure 4.12: Resnet50 Accuracy

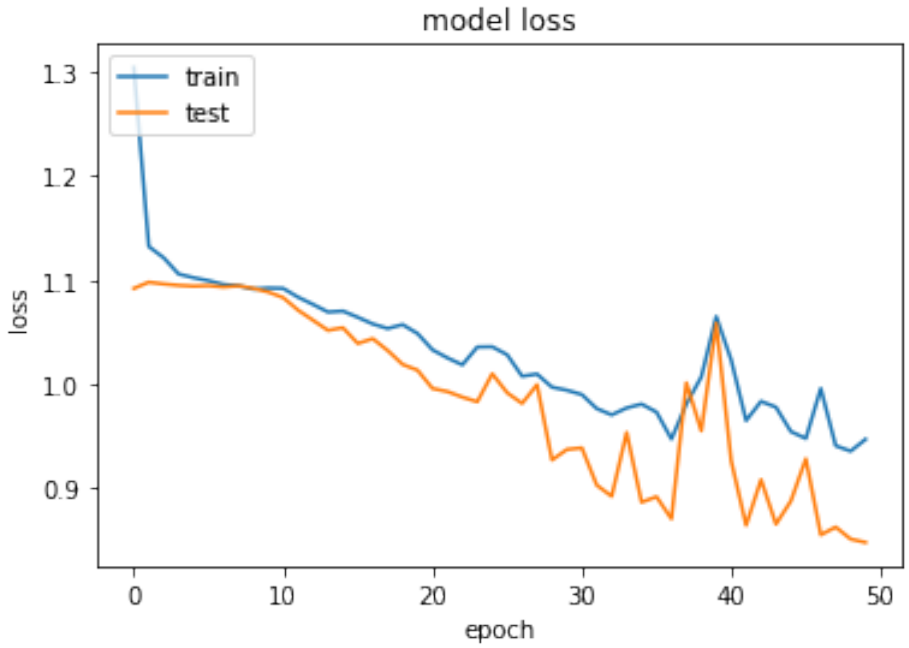


Figure 4.13: Resnet50 Loss

From 4.12 & 4.13 graphs, we can see that the model's accuracy has fluctuated throughout the epochs and the loss graph also fluctuates a lot in the whole graph regions which indicates our model structure does not work well with this pre-trained model.

4.5.2 Precision, Recall & F1 Score Interpretation:

Precision:

Kick	Punch	Slap
0.73	0.74	0.60
ResNet50 Precision Accuracy: 0.68		

Table 4.7: ResNet50 Precision

This table indicates that our model does not give desired output with this pre-trained model.

F1 Score:

Kick	Punch	Slap
0.70	0.74	0.60
ResNet50 F1 Score Accuracy: 0.68		

Table 4.8: ResNet50 F1 Score

F1 score also indicates the same information as the precision table.

Recall:

Kick	Punch	Slap
0.67	0.60	0.77
ResNet50 Recall Accuracy: 0.68		

Table 4.9: ResNet50 Recall

Recall table does not indicate any difference between precision and recall.

4.5.3 Confusion Matrix:

Confusion Matrix - Kick

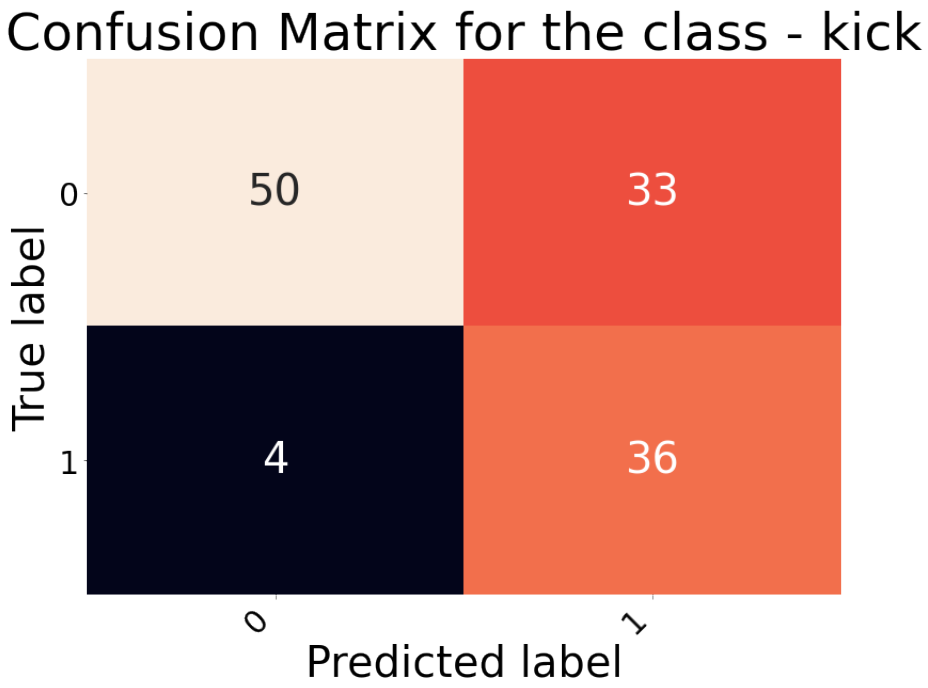


Figure 4.14: ResNet50 Confusion Matrix - Kick

From the 3 confusion matrix, we can see the number of false-positives and the number of false-negatives is higher than VGG16 and InceptionV3. So, our model does not give desired output with this pre-trained model.

Confusion Matrix - Punch

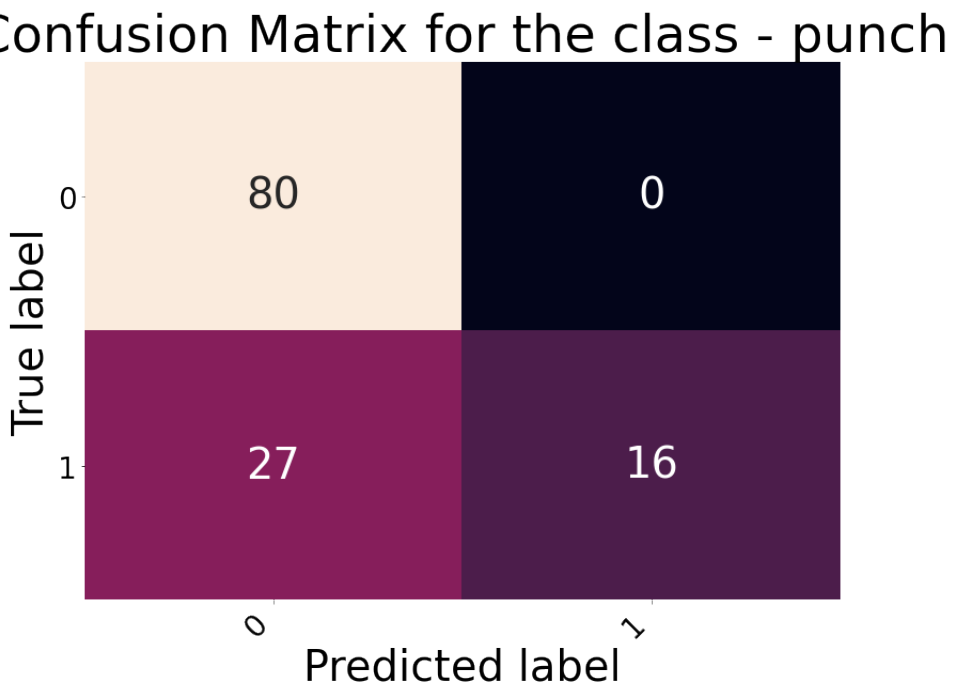


Figure 4.15: ResNet50 Confusion Matrix - Punch

Confusion Matrix - Slap

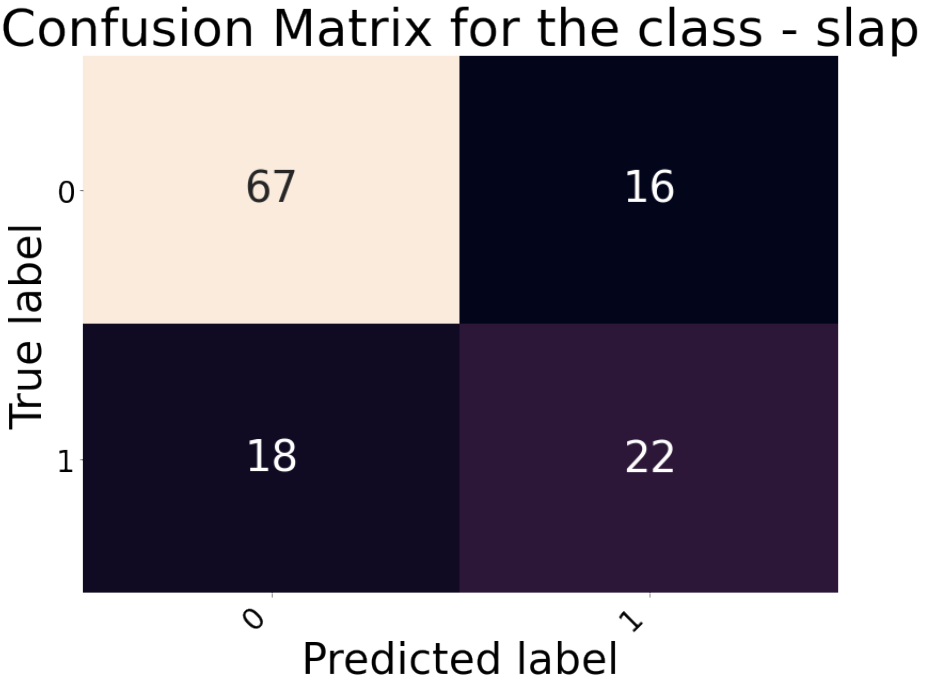


Figure 4.16: ResNet50 Confusion Matrix - Slap

4.6 MobileNet-V2 Implementation and Result

4.6.1 MobileNetV2 Accuracy and Loss:

Using MobileNet V2 as pre-trained model with 50 epochs and batch size 128 we have an accuracy of 92.68%

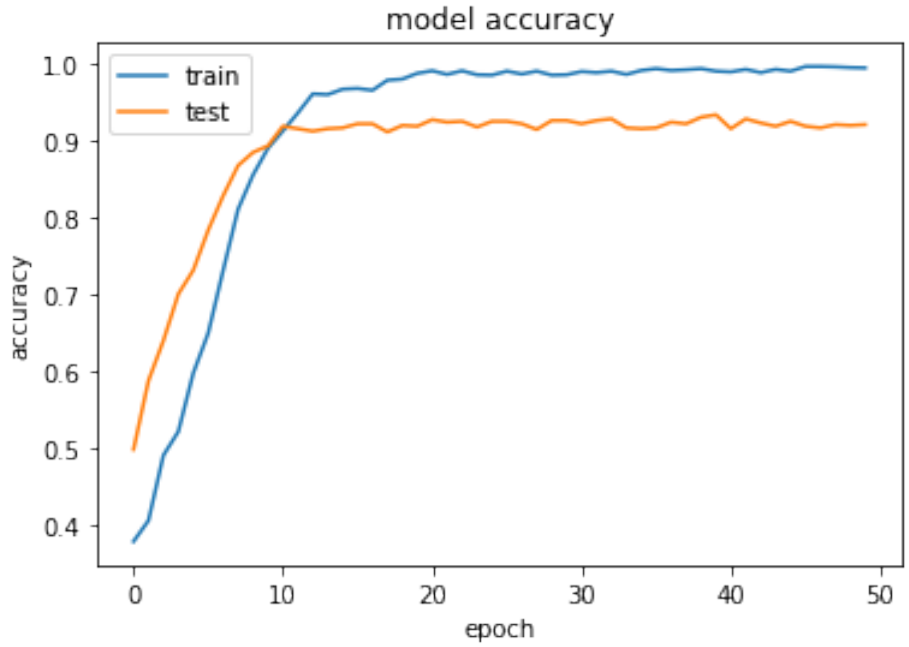


Figure 4.17: MobileNet V2 Accuracy

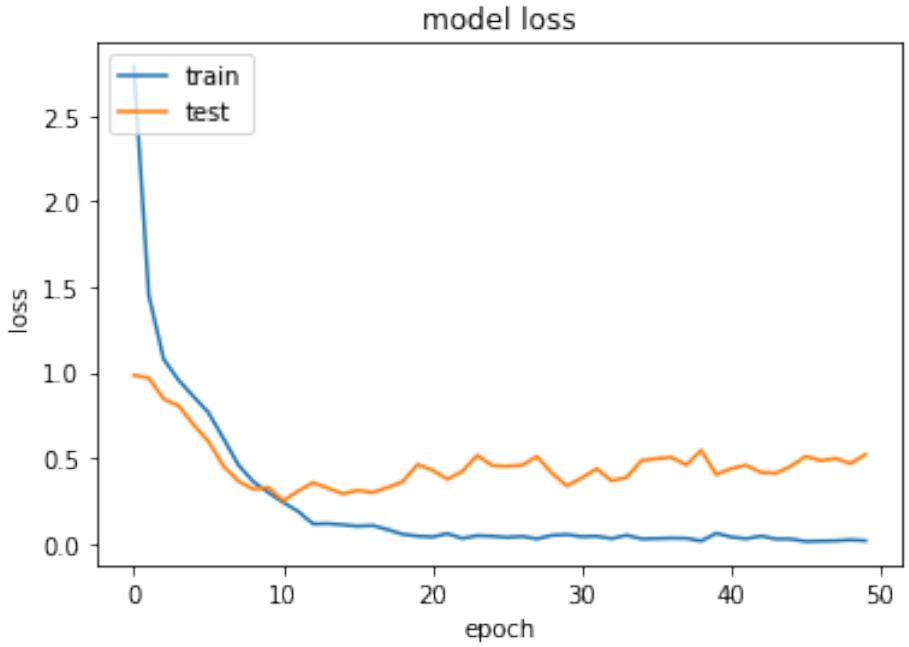


Figure 4.18: MobileNet V2 Loss

From 4.17 & 4.18 graphs,we can see that, model's accuracy has become more linear after 90 which indicates accuracy is between 90 to 95 and the trains' loss graph has

become more linear after 10 to 20 epoch where test loss fluctuate in certain regions from that we can come to a conclusion that we have come to a conclusion that it has also reached a point from where it will not change drastically.

4.6.2 Precision, Recall & F1 Score Interpretation:

Precision:

Kick	Punch	Slap
0.92	0.88	0.97
MobileNet V2 Precision Accuracy: 0.92		

Table 4.10: MobileNet V2 Precision

Values from the precision table represent the good performance of our model.

F1 Score:

Kick	Punch	Slap
0.94	0.91	0.93
MobileNet V2 F1 Score Accuracy: 0.92		

Table 4.11: MobileNet V2 F1 Score

The F1 score table indicates good performance of our model.

Recall:

Kick	Punch	Slap
0.95	0.93	0.90
MobileNet V2 Recall Accuracy: 0.92		

Table 4.12: MobileNet V2 Recall

Recall table indicates good performance of our model.

4.6.3 Confusion Matrix:

From the 3 confusion matrix, we can see the number of false-positives and the number of false-negatives is very little compared to the correct output.

Confusion Matrix - Kick

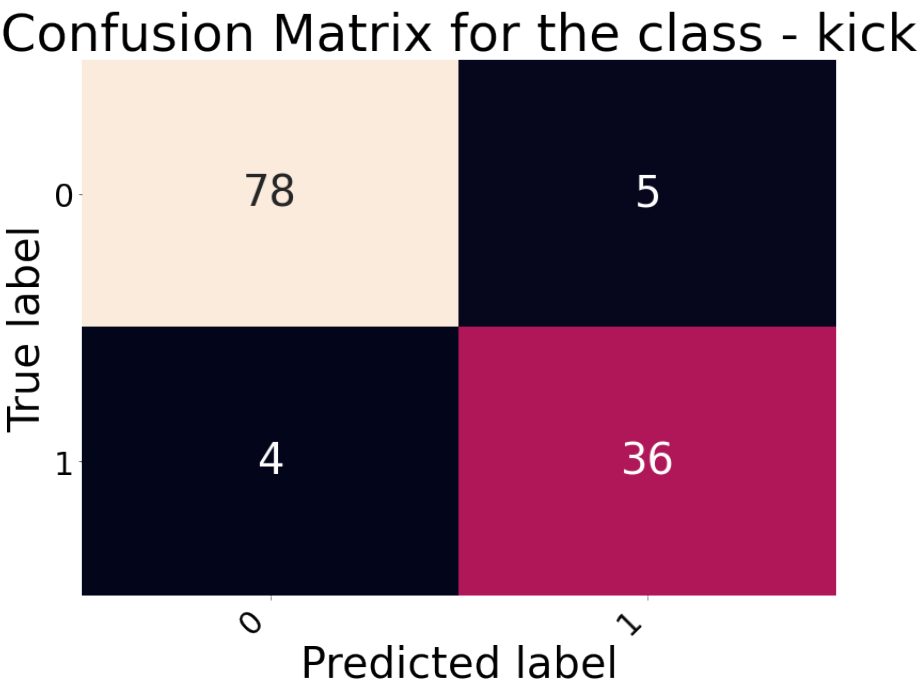


Figure 4.19: MobileNet V2 Confusion Matrix - Kick

Confusion Matrix - Punch

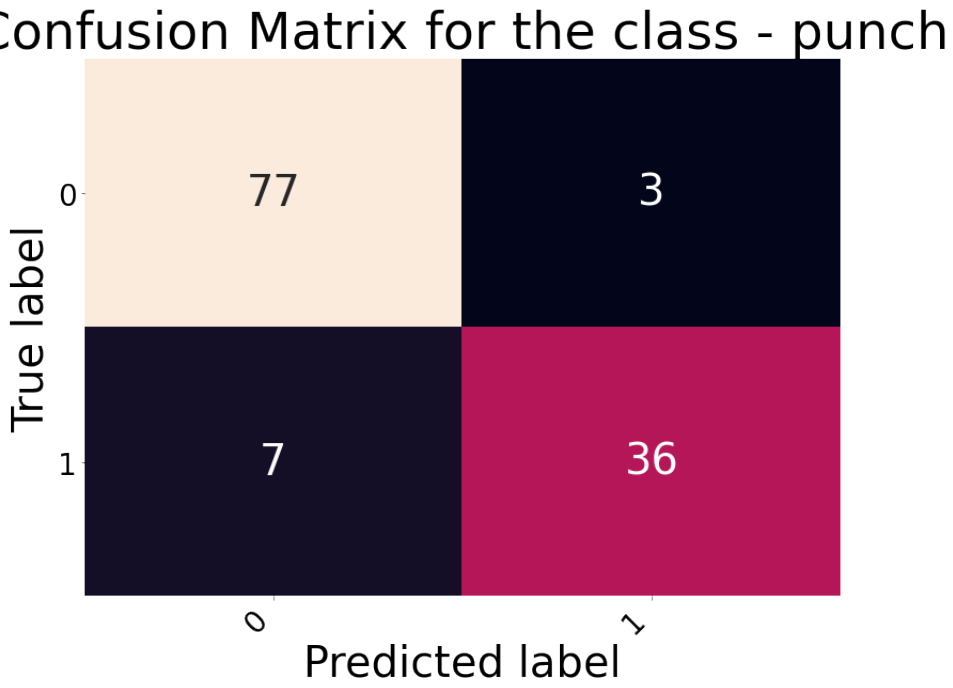


Figure 4.20: MobileNet V2 Confusion Matrix - Punch

Confusion Matrix - Slap

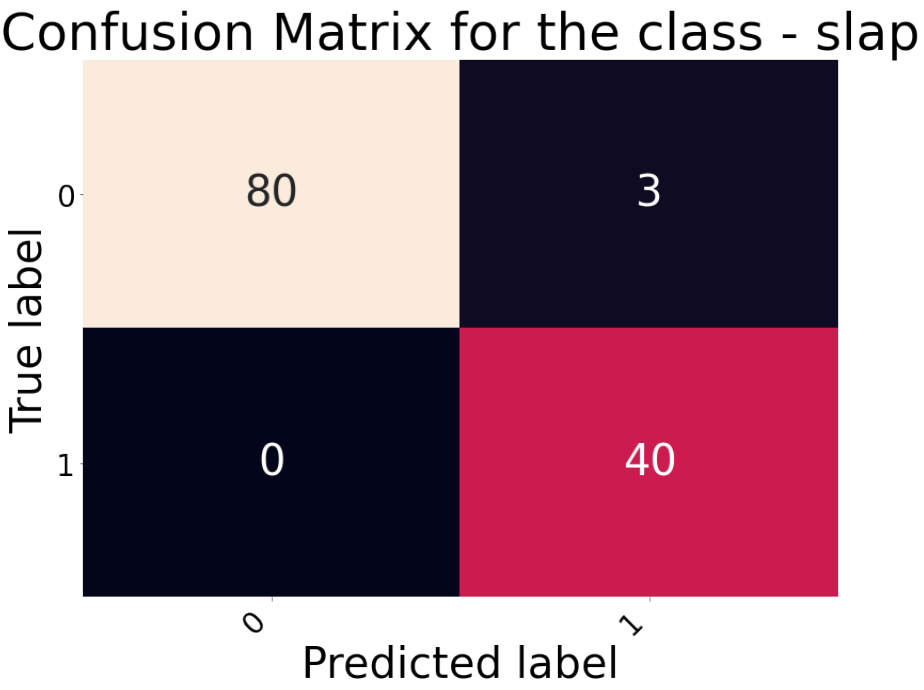


Figure 4.21: MobileNet V2 Confusion Matrix - Slap

4.7 Comparison between different Models

We have used four different image classification pre-trained models.

Model	Precision	F-1 Score	Recall
VGG-16	92	92	92
Inception V-3	88	88	88
ResNet50	68	68	68
MobileNet V-2	92	92	92

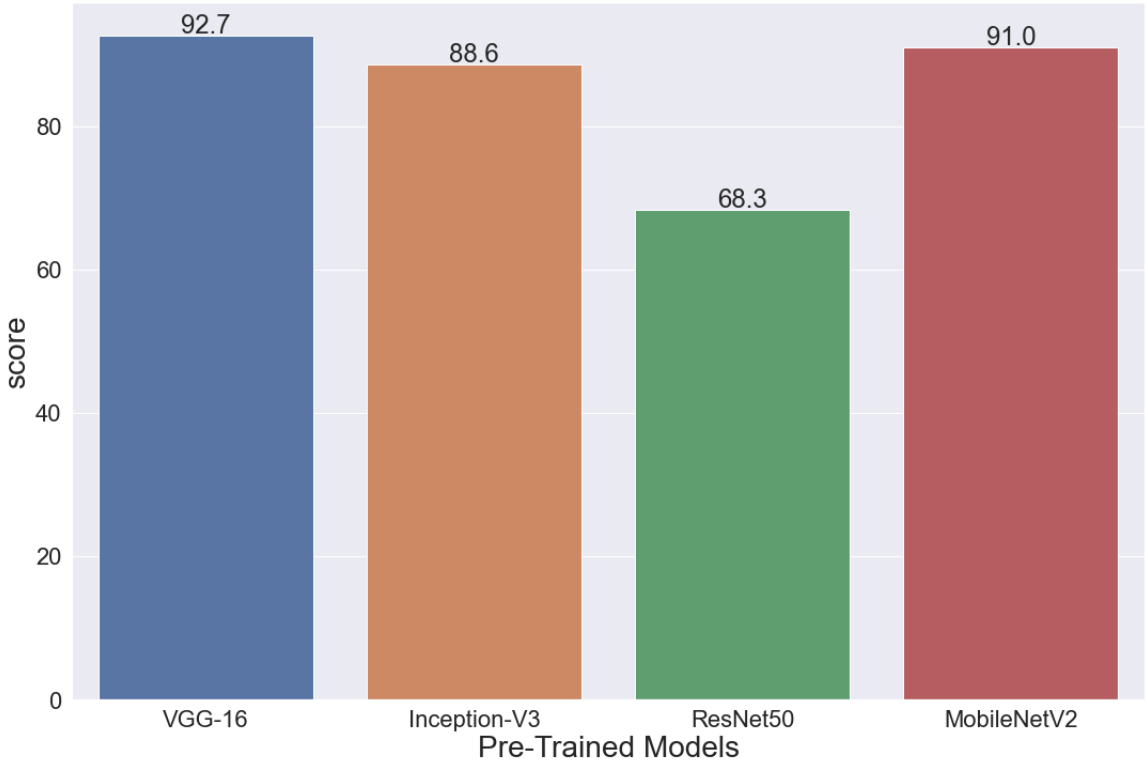


Figure 4.22: Pre-Trained Model's Score

Among the pre-trained models VGG-16's accuracy score was the highest which was 92.7%. With 91% MobileNet V-2 also performed better than the other models. Resnet50 however had the lowest score.

After comparing all the models we can state that VGG-16 and MobileNet V-2 were more stable models with better accuracy.

Chapter 5

Conclusion

Our experiment model, which we have presented, will detect physical bullying from the video footage from a large dataset to find out the violent activities that happened with victims. Though the background noise is a big challenge, it will be able to perform better with high maintenance. It will help detect the bullying activity, which will help the security organizations to take actions against physical bullying, which can turn into big crime. Our model can detect bullying with the existing dataset, and we will make more progress to our model. In the future, we will generate more datasets, and we will also use other machine learning algorithms such as decision tree classifiers, adaboost classifiers, k neighbors classifiers, random forest classifiers, etc to improve our model to work more efficiently.

Bibliography

[1] A. Aljuhani, "Going deeper with convolutions," 1989.

[2] Y. Gong, W. Wang, S. Jiang, Q. Huang, and W. Gao, "Detecting violent scenes in movies by auditory and visual cues," in *Pacific-Rim Conference on Multimedia*, Springer, 2008, pp. 317–326.

[3] Y. Bengio, *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[4] T. Giannakopoulos, A. Makris, D. Kosmopoulos, S. Perantonis, and S. Theodoridis, "Audio-visual fusion for detecting violent scenes in videos," in *Hellenic conference on artificial intelligence*, Springer, 2010, pp. 91–100.

[5] E. B. Nievas, O. D. Suarez, G. B. Garcıa, and R. Sukthankar, "Violence detection in video using computer vision techniques," in *International conference on Computer analysis of images and patterns*, Springer, 2011, pp. 332–339.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] E. Y. Fu, H. V. Leong, G. Ngai, and S. Chan, "Automatic fight detection based on motion analysis," in *2015 IEEE International Symposium on Multimedia (ISM)*, IEEE, 2015, pp. 57–60.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

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

[13] P. Foini, "Video understanding: Fighting scenes recognition,"