

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
**JNANA SANGAMA, BELAGAVI – 590 018, KARNATAKA**



**A Project Report  
on**

***Recognizing Set of Human Activities from Video Dataset using  
Machine Learning***

*Submitted in partial fulfilment of the requirements for the VIII Semester of the degree  
of Bachelor of Engineering in Information Science and Engineering of  
Visvesvaraya Technological University, Belagavi*

Submitted By

<b>Akash Hegde</b>	<b>1RN14IS010</b>
<b>Akshay Sudhir Hulekal</b>	<b>1RN14IS013</b>
<b>Mithun Bharadwaj B V</b>	<b>1RN14IS059</b>
<b>Hitesh B Rao</b>	<b>1RN15IS406</b>

Under the Guidance of

**Dr. M V Sudhamani**

Professor and Head of Department  
Department of ISE  
RNSIT



**Department of Information Science and Engineering**

**RNS Institute of Technology**

**Dr. Vishnuvardhana Road, Rajarajeshwari Nagar Post,  
Channasandra, Bengaluru – 560 098**

**2017-2018**

**RNS INSTITUTE OF TECHNOLOGY**  
**Dr. Vishnuvardhana Road, Rajarajeshwari Nagar Post,**  
**Channasandra, Bengaluru – 560 098**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**



**CERTIFICATE**

Certified that the project entitled *Recognizing Set of Human Activities from Video Dataset using Machine Learning* has been successfully completed by **Akash Hegde (1RN14IS010)**, **Akshay Sudhir Hulekal (1RN14IS013)**, **Mithun Bharadwaj B V (1RN14IS059)** and **Hitesh B Rao (1RN15IS406)**, bonafide students of **RNS Institute of Technology, Bengaluru** in partial fulfilment of the requirements of the VIII Semester for the award of the degree of *Bachelor of Engineering in Information Science and Engineering* of **Visvesvaraya Technological University, Belagavi**, during the academic year **2017 – 2018**. The project report has been approved as it satisfies the academic requirements in respect of project work for the said degree.

---

**Dr. M V Sudhamani**  
Project Guide  
Professor and HoD  
Department of ISE  
RNSIT

---

**Dr. M V Sudhamani**  
Professor and HoD  
Department of ISE  
RNSIT

---

**Dr. M K Venkatesha**  
Principal  
RNSIT

**External Viva**

**Name of Examiners**

1. \_\_\_\_\_
2. \_\_\_\_\_

**Signature with Date**

\_\_\_\_\_  
\_\_\_\_\_

# Declaration

We, **AKASH HEGDE [USN:1RN14IS010]**, **AKSHAY SUDHIR HULEKAL [USN:1RN14IS013]**, **MITHUN BHARADWAJ B V [USN:1RN14IS059]** and **HITESH B RAO [USN:1RN15IS406]**, students of VIII Semester BE, in Information Science and Engineering, RNS Institute of Technology, hereby declare that the Project entitled ***“Recognizing Set of Human Activities from Video Dataset using Machine Learning”*** has been carried out by us and submitted in partial fulfilment of the requirements for the *VIII Semester degree of Bachelor of Engineering in Information Science and Engineering of Visvesvaraya Technological University, Belagavi* during the academic year 2017 – 2018.

Place: Bengaluru

Date:

**AKASH HEGDE (1RN14IS010)**

**AKSHAY SUDHIR HULEKAL (1RN14IS013)**

**MITHUN BHARADWAJ B V (1RN14IS059)**

**HITESH B RAO (1RN15IS406)**

# Abstract

Human Activity Recognition is one of the most advanced and important concept in recent times for the purpose of usage in various applications. The ability to recognize activities is being performed based on a pre-defined dataset after employing machine learning and deep learning techniques, and is a great leap towards creating better machines for the future. There have been numerous attempts at doing these kinds of projects ever since the 1980s and the success of benchmarked and prototype systems always encourages researchers and students alike, to further study the concepts behind activity recognition, machine learning and deep learning.

This project showcases a classifier built on the KTH (Kungliga Tekniska Högskolan) University's dataset which consists of six day-to-day human activities. An attempt is made to first convert the videos of the dataset into frames using FFmpeg and OpenCV and then, build a sequential model that is able to recognize these activities. A combination of Convolutional Neural Networks (CNNs) with Long-Short Term Memory (LSTM) units and Convolutional Neural Networks (CNNs) with Gated Recurrent Units (GRUs), are used in building a layered architecture that is batch normalized to ensure that the network learns and classifies the activities as efficiently as possible.

The performance of FFmpeg and OpenCV is compared with respect to time taken to convert the videos to frames and the total file sizes of the converted frames. Here, it is observed that FFmpeg fares better than OpenCV, in terms of time taken for conversion of videos to frames and memory consumed. Also, a comparative analysis is made between LSTM and GRU which are used in combination with CNN to build the classifier to perform human activity recognition. It can be concluded that CNN combined with LSTM gives better accuracy for the entire dataset, when compared to CNN combined with GRU. Finally, the results obtained are analyzed with respect to performance and discussed. This work also outlines the scope for future enhancements and upgrades to the system, in order to better classify human activities.

# Acknowledgment

A project work is a job of great enormity and it cannot be accomplished by an individual all by them. Hence, we are grateful to a number of individuals whose professional guidance, assistance and encouragement have made it a pleasant experience to undertake this project work.

We take this opportunity to thank our Chairman, **Dr. R N Shetty** and the **Management of RNS Institute of Technology** for providing such a healthy environment for the successful completion of project work.

We would like to express our gratitude to our Director, **Dr. H N Shivashankar** and our Principal, **Dr. M K Venkatesha** for their support and encouragement.

It gives us immense pleasure in expressing gratitude to our respected project guide, **Dr. M V Sudhamani**, Professor and Head of Department, Department of ISE for her constant support, guidance and encouragement throughout the project work.

We would also like to mention special thanks to the Project Coordinators, **Mrs. Kusuma S**, Assistant Professor and **Mr. Manoj Kumar H**, Assistant Professor, Department of ISE and all other teaching and non-teaching staff of Information Science and Engineering Department who have directly or indirectly helped us in the completion of the project work.

We would also like to thank the **Karnataka State Council for Science and Technology (KSCST)** for providing financial support to the project work, through the Student Project Programme (SPP).

We would hereby acknowledge and thank our family and friends for their unfailing moral support and encouragement.

AKASH HEGDE  
AKSHAY SUDHIR HULEKAL  
MITHUN BHARADWAJ B V  
HITESH B RAO

# Contents

<b>Certificate</b>	
<b>Abstract</b>	<b>i</b>
<b>Acknowledgment</b>	<b>ii</b>
<b>Contents</b>	<b>iii</b>
<b>List of Tables</b>	<b>vi</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Abbreviations</b>	<b>viii</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Background	1
1.2 Existing System	2
1.3 Proposed System	3
<b>2. Literature Review</b>	<b>4</b>
<b>3. Analysis</b>	<b>9</b>
3.1 Introduction	9
3.2 Software Requirement Specification	9
3.2.1 Introduction	9
3.2.1.1 Purpose	10
3.2.1.2 Scope	10
3.2.1.3 Definitions, Acronyms and Abbreviations	11
3.2.2 General Description	14
3.2.2.1 Product Perspective	14
3.2.2.2 Product Functions	14
3.2.2.3 User Characteristics	15
3.2.2.4 Assumptions and Dependencies	15
3.2.3 Requirements	15
3.2.3.1 Functional Requirements	15
3.2.3.2 Non-functional Requirements	16
3.2.3.3 Software Requirements	17
3.2.3.4 Hardware Requirements	17
<b>4. Preliminary Design</b>	<b>18</b>
4.1 Introduction	18

4.2 System Architecture	19
4.2.1 Conversion of Videos to Frames	19
4.2.2 Building the Sequential Model	22
<b>5. Detailed Design</b>	<b>23</b>
5.1 High-Level Design	23
5.1.1 Design Consideration	23
5.1.1.1 Assumptions and Dependencies	23
5.1.1.2 Mode of Operation of the System	24
5.1.1.3 User Operations	24
5.1.2 Data Flow Diagram	24
5.2 Low-Level Design	26
5.2.1 Use case Diagram	26
5.2.2 Sequence Diagram	27
5.2.3 Class Diagram	28
<b>6. Implementation Details</b>	<b>31</b>
6.1 Introduction	31
6.2 Overview of System Implementation	31
6.2.1 Usability Aspect	31
6.2.2 Technical Aspect	32
6.2.2.1 Dataset	32
6.2.2.2 Sequential Model	32
6.3 Implementation Support	34
6.3.1 Installing Python	34
6.3.2 Installing PyCharm Community Edition	35
6.3.3 Installing TensorFlow	35
6.3.4 Installing Keras	36
6.3.5 Installing NumPy	36
6.3.6 Installing FFmpeg and OpenCV	36
6.3.7 Installing Matplotlib, Graphviz and Pydot	37
6.3.8 Installing h5py	38
6.3.9 Installing Django	38
6.4 Pseudocodes	38
6.5 Front-End Design	45
<b>7. Testing</b>	<b>46</b>
7.1 Introduction	46

7.2 Levels of Testing	47
7.2.1 Unit Testing	47
7.2.1.1 User Input	47
7.2.1.2 Error Handling	47
7.2.2 Integration Testing	48
7.2.3 System Testing	49
7.2.4 Validation Testing	50
7.2.5 User Acceptance Testing	50
<b>8. Discussion of Results</b>	<b>51</b>
8.1 Log Times of FFmpeg versus OpenCV	51
8.2 Log Times and Accuracies of LSTM and GRU	52
8.2.1 Performance Measures	52
8.2.1.1 Accuracy	52
8.2.1.2 Confusion Matrix	53
8.2.2 LSTM versus GRU	53
<b>9. Conclusions and Future Work</b>	<b>60</b>
<b>References</b>	<b>62</b>



# List of Tables

Table 7.1	Test Cases for Unit Testing of Frame Conversion	47
Table 7.2	Test Cases for Unit Testing of Sequential Model	48
Table 7.3	Test Cases for Integration Testing	49
Table 7.4	Test Cases for System Testing	50
Table 8.1	Log Times of FFmpeg vs. OpenCV	51
Table 8.2	Comparison of Build Model Times of LSTM and GRU	54
Table 8.3	Comparison of Load Data Times of LSTM and GRU	55
Table 8.4	Comparison of Training Times of LSTM and GRU	55
Table 8.5	Comparison of Testing Times of LSTM and GRU	56
Table 8.6	Comparison of Accuracies of LSTM and GRU	56

# List of Figures

Figure 4.1	System Architecture	18
Figure 4.2	Directory Structure of the KTH Dataset	20
Figure 4.3	Directory Structure of “boxing” Activity	20
Figure 4.4	Directory Structure of “handclapping” Activity	20
Figure 4.5	Directory Structure of “handwaving” Activity	21
Figure 4.6	Directory Structure of “jogging” Activity	21
Figure 4.7	Directory Structure of “running” Activity	21
Figure 4.8	Directory Structure of “walking” Activity	21
Figure 4.9	Structure of Sequential Model	22
Figure 5.1	Data Flow Diagram of the System	25
Figure 5.2	Use case Diagram of the System	26
Figure 5.3	Sequence Diagram of the System	28
Figure 5.4	Class Diagram of the System	29
Figure 5.5	User Class	29
Figure 5.6	Model Class	30
Figure 5.7	Dataset Class	30
Figure 6.1	CNN-LSTM Sequential Model	33
Figure 6.2	CNN-GRU Sequential Model	34
Figure 6.3	Pseudocode to Convert Videos to Frames using FFmpeg	39
Figure 6.4	Pseudocode to Convert Videos to Frames using OpenCV	40
Figure 6.5	Pseudocode to Load Data	41
Figure 6.6	Pseudocode to Build Model and Plot Model Architecture	42
Figure 6.7	Pseudocode to Train & Test Model, Calculate & Plot Confusion Matrix, Calculate Accuracy and Save Parameters & Model Architecture	43
Figure 6.8	Pseudocode to Display Front-End Views	44
Figure 6.9	Front-End to Receive Inputs	45
Figure 8.1	Confusion Matrix Definition	53
Figure 8.2	Snapshot of Highest Accuracy Obtained with LSTM for the Entire Dataset	57
Figure 8.3	Snapshot of Highest Accuracy Obtained with GRU for the Entire Dataset	57
Figure 8.4	Sample Confusion Matrix Output for CNN-LSTM Combination	58
Figure 8.5	Sample Confusion Matrix Output for CNN-GRU Combination	59

# List of Abbreviations

ADL	Activities of Daily Living
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AVI	Audio Video Interleave
CCTV	Closed-Circuit Television
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DFD	Data Flow Diagram
DNN	Deep Neural Network
DSF	Django Software Foundation
FN	False Negative
FP	False Positive
FPS	Frames Per Second
GB	Gigabyte
GHz	Gigahertz
GPU	Graphics Processing Unit
Graphviz	Graph Visualization Software
GRU	Gated Recurrent Unit
GRURNTN	Gated Recurrent Unit Recurrent Neural Tensor Network
GUI	Graphical User Interface
HDD	Hard Disk Drive
HDF5	Hierarchical Data Format 5
HLD	High-Level Design
HMDB	Human Motion Database
I&T	Integration & Testing
IDE	Integrated Development Environment
IMDB	Internet Movies Database
ISO	International Standards Organization
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
KTH	Kungliga Tekniska Högskolan

LSTM	Long Short-Term Memory
LSTM-RNTN	Long Short-Term Memory Recurrent Neural Tensor Network
MLP	Multi-Layer Perceptron
MNIST	Modified National Institute of Standards and Technology
MVT	Model-View-Template
NFR	Non-Functional Requirement
ONEIROs	Open-ended Neuro-Electronic Intelligent Robot Operating System
OpenCV	Open-source Computer Vision
OS	Operating System
RAM	Random Access Memory
ReLU	Recurrent Layered Unit
RGB	Red Green Blue
RNN	Recurrent Neural Network
scLSTM	Saliency-Aware 3-Dimensional Convolutional Neural Network with Long Short-Term Memory
SIANN	Shift-Invariant Artificial Neural Network
SME	Subject-Matter Expert
SMPTE	Society of Motion Picture and Television Engineers
SRS	Software Requirement Specification
STFL	Spatio-Temporal Feature Learning
STIP	Spatio-Temporal Interest Point
SVM	Support Vector Machine
TE	Training Epoch
TN	True Negative
TP	True Positive
TPU	Tensor Processing Unit
UAT	User Acceptance Testing
UCF	University of Central Florida
UML	Unified Modeling Language
UPS	Uninterrupted Power Supply
VCS	Version Control System

# Chapter 1

## Introduction

**Activity recognition** aims to recognize the actions and goals of one or more agents from a series of observations on the agents' actions and the environmental conditions. Since 1980s, this research field has captured the attention of several computer science communities due to its strength in providing personalized support for many different applications and its connection to many different fields of study such as medicine, human-computer interaction, or sociology.

Due to its many-faceted nature, different fields may refer to activity recognition as plan recognition, goal recognition, intent recognition, behaviour recognition, location estimation and location-based services.

### 1.1 Background

Computers are getting better at solving some very complex problems like understanding an image due to the advances in computer vision. Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analysing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems. Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, and image restoration.

Models are being made wherein, if an image is given to the model, it can predict what the image is about, or it can detect whether a particular object is present in the image or not. These models are known as neural networks (or artificial neural networks) which are inspired by the structure and functionality of a human brain. Deep learning, a subfield of Machine Learning is the study of these neural networks and over the time, a number of variations of these networks have been implemented for a variety of different problems. This project work uses deep learning for Human Activity Recognition - given a set of videos from a dataset, effectively train a deep network based on the dataset and extract a suitable representation for video-native tasks.

## 1.2 Existing System

The issue of human activity recognition is a very important problem in computer vision. There have been many approaches and datasets used to conduct human activity recognition. The recent technological advancement in deep learning approaches has provided countless possibilities in solving this problem. The part of the problem that makes activity recognition unpredictable is the innumerable ways in which humans think in order to accomplish a given task. For this, there is a need for huge number of video instances in the form of a dataset. This is an endless and laborious task, but there is always a particular pattern in which events occur and a basic set of rules that have to be followed. An attempt is made to extract these loopholes and provide favourable results.

Datasets play a vital role in solving this paramount problem. Not many clean and big enough datasets are available for these tasks that fairly represent real-world conditions. There are a number of human activity video datasets that are used to conduct human activity recognition –

- Weizmann dataset – consists of 90 videos shot on a static camera, each of resolution 180x144 pixels. Evaluation is done by leave-one-out cross validation and the evaluation metric is based on multi-class recognition accuracy. It contains homogeneous outdoor backgrounds in all its videos.
- Charades dataset – consists of approximately 10000 videos that are crowd-sourced and show common household activities with high scene variability. Scaling of this dataset is good and can be a promising dataset for future experiments and explorations.
- UCF-Sports – consists of 182 videos shot on a static camera, each of resolution 720x480 pixels. These are actions collected from various sports in broadcast television channels.
- HMDB51 – consists of 6849 videos, each of resolution 320x240 pixels. These videos are collected from various sources such as movies, YouTube and other public databases. High diversity is present between the videos as some videos are blurred or with lower quality.

### 1.3 Proposed System

This project work aims to conduct human activity recognition on the KTH Dataset using combined neural network approach. The current video database containing six types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed several times by 25 subjects in four different scenarios: outdoors  $d1$ , outdoors with scale variation  $d2$ , outdoors with different clothes  $d3$  and indoors  $d4$ . Currently the database contains 2391 sequences. All sequences were taken over homogeneous backgrounds with a static camera with 25 fps frame rate. The sequences were down-sampled to the spatial resolution of 160 X 120 pixels and have a length of four seconds in average.

All sequences are stored using AVI file format and are available on-line (DIVX-compressed version). Uncompressed version is available on demand. There are  $25 \times 6 \times 4 = 600$  video files for each combination of 25 subjects, 6 actions and 4 scenarios. Each file contains about four sub-sequences used as a *sequence* in our experiments.

Deep learning approaches are used to perform human activity recognition on this KTH Dataset. A combination of Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) is used to create a model, train, validate and test on the dataset. A comparison is made based on the log times that are calculated for conversion of videos to frames and the usage of a Gated Recurrent Unit (GRU) versus the LSTM network. Validation accuracies are compared with changes in the number of epochs, the training set loaded and the test set loaded.

Using TensorFlow and Keras along with supporting Python libraries, an attempt is made to train a classifier on the KTH Dataset such that the confusion matrix obtained at the end is used to recognize human activities being performed in the random samples taken for testing and validation. The basic prototype version of the project deals with recognizing the human actions from a playable video within the KTH Dataset which is already present. This prototype is only sufficient to recognize the basic human behaviour and actions. An enhancement of this project can be done at the industry level by adopting a similar technique using CCTVs to recognize action patterns in real-time. For example, in accidents that occur without human monitoring, disasters or natural calamities that often go undetected due to human negligence; the proposed machine can be upgraded and used to take appropriate measures to avoid future mishaps.

It can also be used in the industry or an institution to detect criminal activities before they are committed or when being committed, such as theft, harassment, etc., through the usage of activity recognition software embedded in the CCTVs for video surveillance.

## Chapter 2

### Literature Review

Human activity recognition is a widely explored and experimented topic in the field of computer vision. There have been numerous researchers who have conducted countless experiments at the expense of compute resources and time. The papers related to human activity recognition and the KTH Dataset have been surveyed and reviewed throughout the conduction of this project.

Shikhar Shrestha of Stanford University, California has stated that human activity recognition is a very important problem in computer vision that is largely unsolved. The researcher further states that while recent advances in areas such as deep learning have given us great results on image related tasks; it is still unclear as to what a good feature representation is for recognizing activities from videos. A large part of the problem is that not many clean and big enough data sets are available for this task that fairly represents real-world conditions. Recently a new dataset has been made available to the research community by Allen Institute that has enough content to effectively train a deep network and extract a suitable representation for video-native tasks. This work implements state-of-the-art deep video feature extraction on this dataset and then trains a classifier to perform Activity Recognition. The results are then compared and benchmarked. Fusion based architecture is given importance based on convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are found to be more accurate than local feature descriptors in some situations [1].

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar and Li Fei-Fei have published a paper that illustrates the large-scale video classification using convolutional neural networks (CNNs). An extensive empirical evaluation of CNNs on large-scale video classification is done using a dataset that contains 1 million YouTube videos belonging to 487 classes. Multiple approaches for extending the connectivity of a CNN in time domain to take advantage of local spatio-temporal information are studied and a multiresolution and clear architecture acting as a way to speed up the process of training is suggested. Significant performance improvements are observed in the spatio-temporal networks when compared to strong feature-based baselines. A modest performance improvement is observed when the spatio-temporal networks are compared to single-frame models. Further, the researchers study the generalization performance of their best spatio-temporal network model by retraining the top layers on the UCF-101 Action Recognition Dataset and observe the significant performance improvements compared to the UCF-101 baselines [2].



Christian Schüldt, Ivan Laptev and Barbara Caputo demonstrate that activity recognition can be performed on the KTH Dataset using local measurements in terms of spatio-temporal interest points (STIPs), which are known as local features. Such features capture local motion events in video and can be adapted to the size, the frequency and the velocity of moving patterns, hence, resulting in video representations that are stable with respect to corresponding transformations. They construct video representations in terms of local space-time features and integrate such representations with SVM classification schemes for recognition. They state that local features give robust recognition performance in scenes with complex non-stationary backgrounds and the spatial and the temporal relations between features provide additional cues that could be used to improve the results of recognition [3].

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani and Manohar Paluri have proposed an effective approach for spatio-temporal feature learning using deep 3-dimensional convolutional networks (3D ConvNets) that are trained on a large-scale supervised video dataset. They have found that 3D ConvNets are more suitable for spatio-temporal feature learning compared to 2D ConvNets, a homogeneous architecture with small 3x3x3 convolution kernels in all layers is among the best performing architectures for 3D ConvNets. They also state that the 3D ConvNets outperform state-of-the-art methods on four different benchmarks and are comparable with current best methods on two other benchmarks [4].

Nitish Srivastava, Elman Mansimov and Ruslan Salakhutdinov use multilayer long short-term memory (LSTMs) networks to learn representations of video sequences. They propose that their model uses an encoder LSTM to map an input sequence into a fixed length representation. This representation is decoded using single or multiple decoder LSTMs to perform different tasks, such as reconstructing the input sequence, or predicting the future sequence. They experiment with two kinds of input sequences – patches of image pixels and high-level representations (percepts) of video frames extracted using a pre-trained convolutional net. They explore different design choices such as whether the decoder LSTMs should condition on the generated output. Finally, they analyse the outputs of the model qualitatively to see how well the model can extrapolate the learned video representation into the future and into the past [5].

Yang Qin, Lingfei Mo, Jing Ye and Zhening Du investigate a general feature combination method for relatively new 3D CNNs and LSTMs fusion model in human activity recognition. All the features used in this combination method are from human activity videos without manually extracting features or any prior knowledge, and the model has good generalization performance. Through extracting multichannel features of the motion optical flow vector, grayscale and body edge,

putting them to 3D convolutional neural network, and processing time characteristics within Long-Short Term Memory neural network, the recognition rate of the model rises greatly. The experiment selects KTH dataset as the data source. The model based on RGB is used to compare with the model based on multi-channel features. It shows that multi-channel features can improve recognition accuracy rate obviously, and have great robustness in different scenes, which proves that it is an efficient feature combination method fitted 3D CNNs and LSTMs [6].

Andros Tjandra, Sakriani Sakti, Ruli Manurung, Mirna Adriani and Satoshi Nakamura have proposed a novel RNN architecture that combines the concepts of gating mechanism and the tensor product into a single model. By combining these two concepts into a single RNN, their proposed models learn long-term dependencies by modeling with gating units and obtain more expressive and direct interaction between input and hidden layers using a tensor product on 3-dimensional array (tensor) weight parameters. They use Long Short Term Memory (LSTM) RNN and Gated Recurrent Unit (GRU) RNN and combine them with a tensor product inside their formulations. Their proposed RNNs, which are called a Long-Short Term Memory Recurrent Neural Tensor Network (LSTMRNTN) and Gated Recurrent Unit Recurrent Neural Tensor Network (GRURNTN), are made by combining the LSTM and GRU RNN models with the tensor product. Experiments were conducted with their proposed models on word-level and character-level language modeling tasks and revealed that the proposed models significantly improved their performance compared to the baseline models [7].

Xuanhan Wang, Lianli Gao, Jingkuan Song and Hengtao Shen propose a novel pipeline, Saliency-aware 3D CNN with LSTM (scLSTM), for video action recognition by integrating LSTM with salient-aware deep 3DCNN features on video shots. They say that it is better than the existing model which feeds frame-level CNN sequence features to Long Short-Term Memory (LSTM) model for video activity recognition. This recurrent model based visual recognition pipeline is a natural choice for perceptual problems with time-varying visual input or sequential outputs. However, the above pipeline takes frame-level CNN sequence features as input for LSTM, which may fail to capture the rich motion information from adjacent frames or maybe multiple clips. Furthermore, an activity is conducted by a subject or multiple subjects. It is important to consider attention which allows for salient features, instead of mapping an entire frame into a static representation. They first apply saliency-aware methods to generate saliency-aware videos. Then, they design an end-to-end pipeline by integrating 3D CNN with LSTM, followed by a time series pooling layer and a Softmax layer to predict the activities [8].

Nicolas Ballas, Li Yao, Chris Pal and Aaron Courville propose an approach to learn spatio-temporal features in videos from intermediate visual representations called “percepts” using Gated-Recurrent-Unit Recurrent Networks (GRUs). Their method relies on percepts that are extracted from all levels of a deep convolutional network trained on the large ImageNet dataset. While high-level percepts contain highly discriminative information, they tend to have a low-spatial resolution. Low-level percepts, on the other hand, preserve a higher spatial resolution from which we can model finer motion patterns. Using low-level percepts, however, can lead to high-dimensionality video representations. To mitigate this effect and control the number of parameters, they introduce a variant of the GRU model that leverages the convolution operations to enforce sparse connectivity of the model units and share parameters across the input spatial locations. They empirically validate their approach on both Human Action Recognition and Video Captioning tasks. In particular, they achieve results equivalent to state-of-art on the YouTube2Text dataset using a simpler caption-decoder model and without extra 3D CNN features [9].

Fadwa Al-Azzo, Chunbo Bao, Arwa Mohammed Taqi, Mariofanna Milanova and Nabeel Ghassan propose a model based on 3D deep neural network (3D DNN) to recognize human actions. To classify human actions, the recognition process is implemented under different recording conditions from a surveillance camera. By applying Caffe\_GoogLeNet framework, the 3D DNN was trained with different training epoch values (TEs). The experiments were then evaluated using three different datasets: KTH, Weizmann, and UCF101 with gray and colour resolutions. The results of the experiments demonstrate significantly high performance in the recognition rates by changing the training epoch values (TEs) to accomplish the best classification accuracy with a remarkable short running time. The classification accuracy results of 3D DNN with other state-of-the-art for three datasets are then compared [10].

Wen-Hui Chen, Carlos Andrés Betancourt Baca and Chih-Hao Tou look at the healthcare applications of human activity recognition, such as helping doctors keep track of patient history and analyze patient behaviours in order to make a better diagnosis. An attempt is made to create an accurate activity recognition system dedicated on activities of daily living (ADLs) using deep learning models that can help develop a senior healthcare system. Developing an accurate activity recognition system with sensor readings is a challenging task due to the fact that sensors are subjected to a variety of errors and the nature of human activities is dynamic and uncertainties. Recent studies have shown that machine learning approaches can effectively classify human activities. Sensor readings from accelerometers and gyroscopes is analysed using long short-term memory (LSTM) recurrent neural networks to identify human activities and present a location-aware

approach to improve the recognition accuracy. The location information is obtained from analysing images captured by a wearable camera based on a pre-trained model of the deep convolutional neural networks. With the location information, some unlikely activities can be ruled out, leading to better recognition accuracy [11].

Rahul Dey and Fathi M. Salem present a paper that evaluates three variants of the gated recurrent unit (GRU) in recurrent neural networks (RNNs) by retaining the structure and systematically reducing parameters in the update and reset gates. The three variant GRU models are evaluated on MNIST and IMDB datasets and it can be observed that these GRU-RNN variant models perform as well as the original GRU RNN model while reducing the computational expense. In this comparative study, they simply refer to the three variants as, respectively, GRU1, GRU2, and GRU3 RNNs. Gated RNNs' success is primarily due to the gating network signalling that control how the present input and previous memory are used to update the current activation and produce the current state. These gates have their own sets of weights that are adaptively updated in the learning phase (i.e., the training and evaluation process). While these models empower successful learning in RNNs, they introduce an increase in parameterization through their gate networks. Consequently, there is an added computational expense vis-a-vis the simple RNN model. It is noted that the LSTM RNN employs 3 distinct gate networks while the GRU RNN reduces the gate networks to two [12].

John See and Saimunur Rahman investigate the effects of low video quality in human action recognition from two perspectives: videos that are poorly sampled spatially (low resolution) and temporally (low frame rate), and compressed videos affected by motion blurring and artifacts. A wide variety of approaches have shown to work well against challenging image variations such as appearance, pose and illumination. However, the problem of low video quality remains an unexplored and challenging issue in real world applications. In order to increase the robustness of feature representation under these conditions, the usage of textural features to complement the popular shape and motion features is proposed. Extensive experiments were carried out on two well-known benchmark datasets of contrasting nature: the classic KTH dataset and the large-scale HMDB51 dataset. Results obtained with two popular representation schemes (Bag-of-Words, Fisher Vectors) further validate the effectiveness of the proposed approach [13].

## Chapter 3

# Analysis

The following section describes the software requirement specification of the project. This includes the purpose and scope, the general description and requirements of the project.

### 3.1 Introduction

The proposed project is based on machine learning concepts. Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Deep learning models are loosely related to information processing and communication patterns in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neuronal responses in the brain.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design, where they have produces results comparable to and in some cases superior to human experts.

### 3.2 Software Requirement Specification

The software requirement specification is the description of a software system to be developed. It is a document that enlists enough and necessary requirements that are required for the project development. To derive the requirements, the developer needs to have clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

#### 3.2.1 Introduction

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven project, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can

begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Used appropriately, software requirements specifications can help prevent software project failure.

The Software Requirements Specification (SRS) is a communication tool between stakeholders and software designers. The specific goals of the SRS are:

- Give a precise purpose of the project work with respect to feature extraction from videos.
- Describing the scope of the work, in terms of the concepts of machine learning and deep learning.
- Providing a reference to software designers, using different case diagrams and data flow diagrams.
- Providing a framework for performing testing activities using different input training and testing parameters, and number of epochs.
- Providing a platform to add more features to the system to better classify human activities.

### **3.2.1.1 Purpose**

The main aim of this project is to implement a state-of-the-art video feature extraction on the KTH Dataset and train a model to perform human activity recognition, using deep learning techniques. The purpose of specifying the required software for the project is to give a clear picture of the necessary requirements that are used to build the model and implement the tasks to perform human activity recognition.

### **3.2.1.2 Scope**

The scope of the project deals with concepts of machine learning and deep learning. Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to “learn”, that is, progressively improve performance on a specific task with data, without being explicitly programmed. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. This project employs the technique of building a sequential model with a combination of convolutional neural networks (CNNs) and either long short-term memory (LSTMs) units or gated recurrent units (GRUs).

### 3.2.1.3 Definitions, Acronyms and Abbreviations

This sub-section deals with the various definitions, acronyms and/or abbreviations used in the project. These are all applicable and related to the concepts used in the project.

An **image** is an artifact that depicts visual perception, for example, a photo or a two-dimensional picture that has a similar appearance to some subject – usually a physical object or person, thus providing a depiction of it. An image can be in greyscale or can contain RGB channels. In this project, the conversion of each video in the dataset is done to frames which are static and still greyscale images of a person performing an activity.

A **frame** is one of the many still images which compose the complete moving picture. When the moving picture is displayed, each frame is flashed on the screen for a short time and then immediately replaced by the next one. Persistence of vision blends the frames together, producing the illusion of a moving image. In this project, frames are the key requirement in order to determine the activity that is being performed.

**Frame rate** is the frequency (rate) at which consecutive frames appear on a display. It is expressed in terms of Frames Per Second (FPS). In this project, the dataset used consists of videos which all have a constant frame rate of 25 FPS.

A **video** is a series of still images that, when shown on a screen; create the illusion of moving images. This optical illusion causes the audience to perceive continuous motion between separate objects viewed in rapid succession. In this project, the dataset used for human activity recognition consists of 600 videos, each of which is converted into frames and the model is trained and tested on these frames.

In digital imaging, a **pixel** is a physical point in a raster image, or the smallest addressable element in an all points addressable display device. Each pixel is a sample of an original image; more samples typically provide more accurate representations of the original.

**Resolution** of an image or video is the detail that the particular image or video holds. It can be represented in terms of width X height, with the units specified in pixels. In this project, the videos in the dataset and the subsequently generated images/frames have a resolution of 160 X 120 pixels.

**Activity recognition** aims to recognize the actions and goals of one or more agents from a series of observations on the agents' actions and the environmental conditions. Since the 1980s, this



research field has captured the attention of several computer science communities due to its strength in providing personalized support for many different applications and its connection to many different fields of study such as medicine, human-computer interaction, or sociology.

A **model** is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. It is the core data structure used in the project, which can be defined as a way to organize different layers in a sequential approach.

**Machine learning** is a field of computer science that uses statistical techniques to give computer systems the ability to “learn”, that is, progressively improve performance on a specific task with data, without being explicitly programmed. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs.

In machine learning, **feature learning** is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. It is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process.

**Deep learning** is a part of a broader family of machine learning methods, based on learning data representations, as opposed to task-specific algorithms. Deep learning models are loosely related to information processing and communications pattern in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neural responses in the brain.

**Computer vision** is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

**Convolution** is a mathematical operation on two functions to produce a third function, that is typically viewed as a modified version of one of the original functions, giving the integral of the point wise multiplication of the two functions as a function of the amount that one of the original functions is translated.

An **artificial neural network (ANN)** is a computing system based on a collection of connected units or nodes called artificial neurons (a simplified version of biological neurons in an



animal brain). Each connection (a simplified version of a synapse) between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

A **feed forward neural network** is an artificial neural network wherein connections between the units do not form a cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

In computational networks, **activation function** of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be ON (1) or OFF (0), depending on input.

A **multilayer perceptron (MLP)** is a class of feed forward artificial neural network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

In machine learning, a **convolutional neural network (CNN, or ConvNet)** is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal pre-processing. They are also known as shift-invariant or space-invariant artificial neural networks (SIANN), based on their shared weights architecture and translation invariance characteristics. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

A **recurrent neural network (RNN)** is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behaviour for a time sequence. Unlike feed forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

**Long short-term memory (LSTM)** units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feed forward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell.

**Gated recurrent units (GRUs)** are a gating mechanism in recurrent neural networks. Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory. However, GRUs have been shown to exhibit better performance on smaller datasets. They have fewer parameters than LSTM, as they lack an output gate. They combine the forget and input gates into a single "update gate." They also merge the cell state and hidden state. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

### **3.2.2 General Description**

This section describes the general description of the project with respect to product perspective and functions, user characteristics and the assumptions and dependencies involved in the project.

#### **3.2.2.1 Product Perspective**

This project based on human activity recognition is intended to help people understand the basics of deep learning techniques and give an insight into how human activity recognition can be modelled for use in day-to-day life for simple activities.

#### **3.2.2.2 Product Functions**

The code involved in human activity recognition is completely based on open-source resources; hence it can be modified to any required specifications and applications.

The functions of the said code can be given as follows:

- An introduction to deep learning and its techniques used in human activity recognition.

- An understanding of how videos of a dataset can be converted into frames and machine learning concepts can be applied on them.
- Classification of labels in the dataset based on the recognized activity.
- Plotting of the deep learning network model involved in classification.
- Sketching a confusion matrix in order to showcase the difference between predicted and actual label of the classes involved.

### **3.2.2.3 User Characteristics**

The user of this particular model is considered to have a basic understanding of classification and machine learning concepts. Furthermore, the user needs to have the dataset available, either stored locally or with the ability to download it from the necessary website in order to carry out activity recognition.

### **3.2.2.4 Assumptions and Dependencies**

Several assumptions and dependencies can be specified for human activity recognition being carried out in this project. They can be given as follows:

- The dataset is assumed to be complete and there are no missing videos and/or corrupted data.
- The dataset is also assumed to be untouched, i.e., no changes to be made to the dataset once it has been downloaded and stored locally.
- The required technology and software is assumed to have already been installed on the machine running the scripts required to perform human activity recognition.
- The project is dependent on the computer or machine having minimum specifications required to run the particular associated software.

## **3.2.3 Requirements**

This section describes the requirements necessary for the project to work efficiently and produce the required output, given a set of inputs. The requirements can be functional, non-functional, software and hardware requirements.

### **3.2.3.1 Functional Requirements**

A functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behaviour, and outputs.

Here, in this particular project, the functional requirements include the following:

- The system performing human activity recognition must be able to handle the dataset that is used in the project.
- The dataset must be credible, containing a proper collection of specified videos according to the given specifications and not a collection of assorted videos.
- The number of epochs, training and testing parameters must be specified properly and within the given range.
- Erratic behaviour, if any, must be accounted for and required changes have to be made to the system performing human activity recognition.

### **3.2.3.2 Non-functional Requirements**

A non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. NFRs are usually called “quality attributes” of a system.

The NFRs of this project can be given as follows:

- Accessibility – The proposed system must be accessible at all times and must be open-source to all users, without any restrictions.
- Adaptability – The proposed system must be adaptable to further changes made to it in terms of additional parameters or increased complexity.
- Availability – The system must be available at all times and must be running whenever necessary, even if it means having an uninterrupted power supply (UPS) specification.
- Backup – The system and the corresponding model parameters and dataset must be backed up in at least two locations to ensure easy and faster recovery.
- Cost – The system is based on open-source resources, hence the cost must be less.
- Extensibility – The system must be open to addition of new features and carry-forward of customizations at next major version upgrade.
- Performance/platform compatibility – The system must be able to deliver great performance on different platforms.
- Testability – The system must be able to handle the different test cases generated with respect to number of epochs, training and testing set parameters.
- Usability – The system must be simple and must be usable by anyone with a particular interest in machine learning.

### **3.2.3.3 Software Requirements**

Every project needs to have software that have to be installed in order to efficiently run the model and the scripts. This project needs the following software requirements:

- Operating System – Windows 7 Service Pack 1 or above.
- Programming Languages – Python 3.6 and its supporting frameworks and APIs.
- Front-end – Django and supporting libraries.
- Back-end – KTH Dataset, Keras over TensorFlow backend (CPU only) and supporting libraries.
- Script editor – PyCharm Community Edition.
- Supporting technologies – FFmpeg, OpenCV, Graphviz.

### **3.2.3.4 Hardware Requirements**

There is an absolute necessity for hardware that have to be set up in order to run the software necessary to perform HAR. The hardware requirements are given as follows:

- Processor – Multi-core x64-based architecture CPU with at least 2GHz, with an optional GPU to effectively train and test the model, on a compatible motherboard.
- RAM – 4GB of RAM.
- Hard Disk Capacity – At least 100GB to accommodate OS, necessary software requirements, KTH Dataset and backup data.
- Peripherals – Display unit (monitor), compatible keyboard and mouse.

# Preliminary Design

## 4.1 Introduction

Preliminary design involves the description of the system architecture and the necessary diagrams and the models used in the system. The system architecture is defined as the construction of the components required in order to build the model necessary to run the system. In this project, human activity recognition requires the use of multiple systems in order to conduct the project. The main systems necessary are the conversion of videos to frames and building of a sequential model to build the classifier required to classify the activities in the dataset. The main architecture of the system can be given as shown in Figure 4.1.

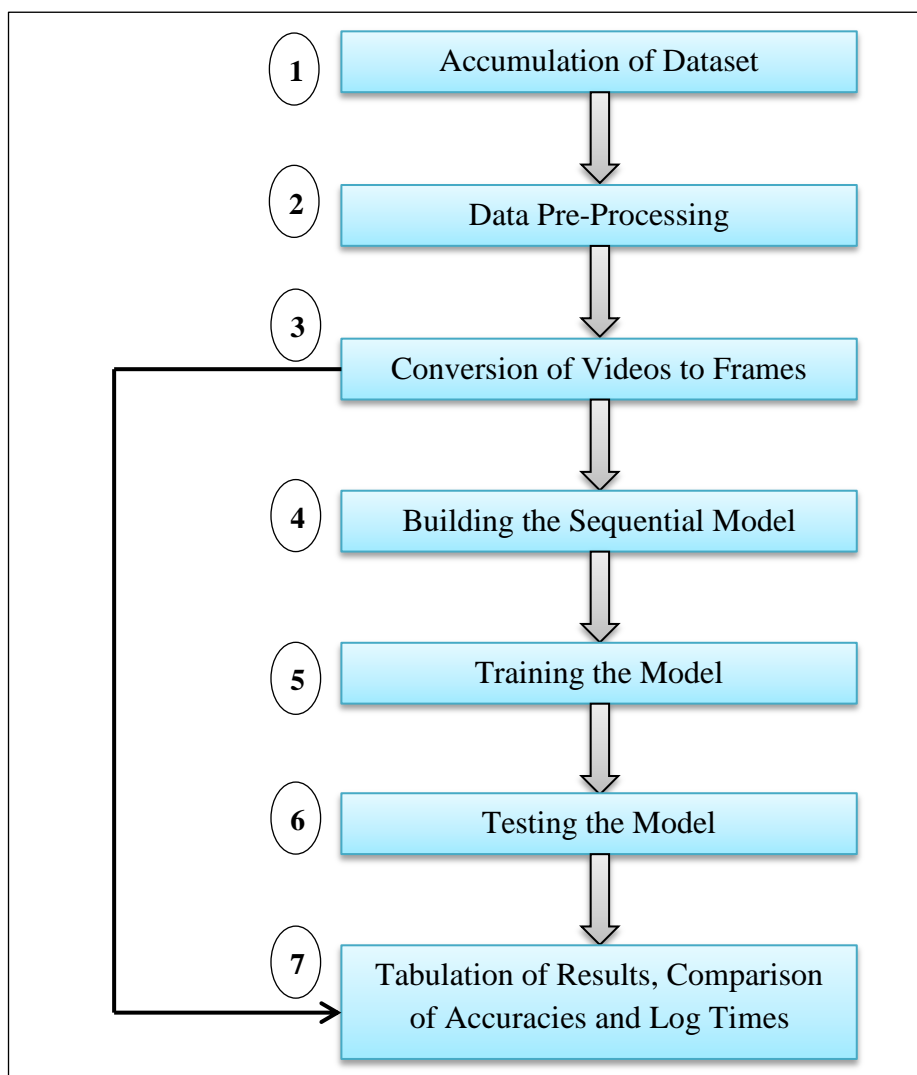


Figure 4.1 System Architecture

## 4.2 System Architecture

This section describes the architecture of the system given in Figure 4.1. In the first module, the accumulation of the dataset is done, from the official website where the KTH Dataset is described by the Kungliga Tekniska Högskolan (KTH) University, Sweden. The second module consists of pre-processing. The obtained dataset is pre-processed by creating a text file, which consists of the sequence numbers of frames in which the activities take place. This is followed by the main module, i.e., the third module, of conversion of videos to frames, which is done with respect to two technologies – FFmpeg and OpenCV. The results of this module are tabulated and compared in the seventh module. The fourth module involves the building of the sequential model based on the combined neural network approach, involving CNN + LSTM, and CNN + GRU. The fifth and sixth modules describe the training and testing of the sequential model with respect to parameters defined by the user. Finally, the seventh module tabulates the results obtained from both conversions of videos to frames and the classification of human activities using the sequential model. Accuracies of the classifiers are compared and the time taken for various experiments are logged.

### 4.2.1 Conversion of Videos to Frames

The first main part of the project deals with conversion of videos to frames. This project aims to conduct human activity recognition on the KTH Dataset using a combined neural network approach. The current video database containing six types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed several times by 25 subjects in four different scenarios: outdoors *d1*, outdoors with scale variation *d2*, outdoors with different clothes *d3* and indoors *d4*. Currently the database contains 2391 sequences. All sequences were taken over homogeneous backgrounds with a static camera with 25 fps frame rate. The sequences were down-sampled to the spatial resolution of 160 X 120 pixels and have a length of four seconds in average. All sequences are stored using AVI file format and are available on-line (DIVX-compressed version). Uncompressed version is available on demand. There are  $25 \times 6 \times 4 = 600$  video files for each combination of 25 subjects, 6 actions and 4 scenarios. Each file contains about four sub-sequences used as a *sequence* in our experiments.

These videos are stored in a folder which consists of six sub-folders, each having the activities as the folder names, i.e., boxing, handclapping, handwaving, jogging, running and walking. These activity folders have 100 videos each, which consists of 25 persons, each captured in 4 different scenarios in the following format: “person” followed by a two digit number referring to the person number that lies between 01 and 25, followed by an underscore, followed by activity name,

followed by an underscore, followed by the scenario number in the format of “d” followed by a number between 1 and 4, followed by an underscore, followed by “uncomp” referring to the uncompressed version of the video and finally the extension of the video, that is “.avi”. For example: “person12\_handclapping\_d3\_uncomp.avi” means that the video selected is of person number 12, performing handclapping activity under scenario number 3 (outdoors with different clothes). The directory structure for the dataset is given as shown in Figure 4.2. Individual directory structures for all six activities are shown in Figures 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8.

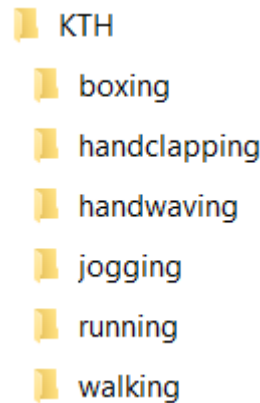


Figure 4.2 Directory Structure of the KTH Dataset

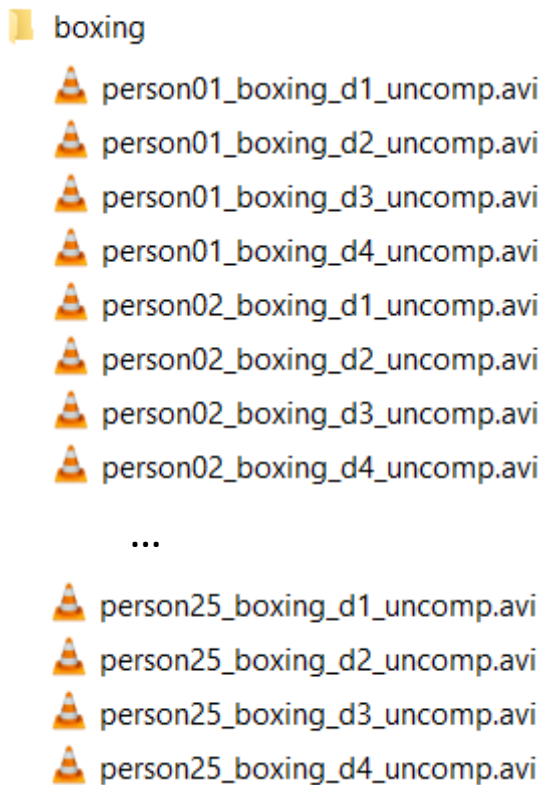


Figure 4.3 Directory Structure of “boxing” Activity

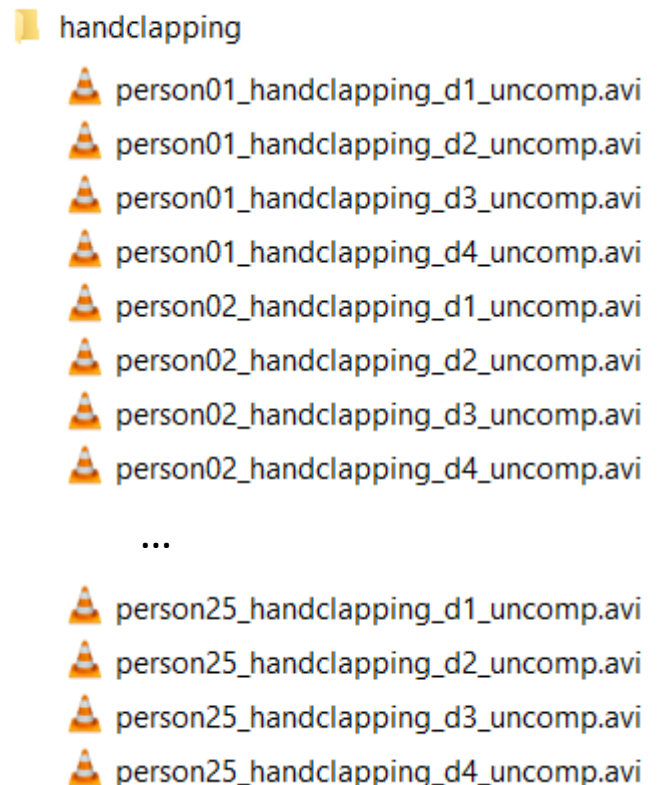


Figure 4.4 Directory Structure of “handclapping” Activity



📁 handwaving

📺 person01\_handwaving\_d1\_uncomp.avi  
📺 person01\_handwaving\_d2\_uncomp.avi  
📺 person01\_handwaving\_d3\_uncomp.avi  
📺 person01\_handwaving\_d4\_uncomp.avi  
📺 person02\_handwaving\_d1\_uncomp.avi  
📺 person02\_handwaving\_d2\_uncomp.avi  
📺 person02\_handwaving\_d3\_uncomp.avi  
📺 person02\_handwaving\_d4\_uncomp.avi

...

📺 person25\_handwaving\_d1\_uncomp.avi  
📺 person25\_handwaving\_d2\_uncomp.avi  
📺 person25\_handwaving\_d3\_uncomp.avi  
📺 person25\_handwaving\_d4\_uncomp.avi

Figure 4.5 Directory Structure of “handwaving” Activity

📁 jogging

📺 person01\_jogging\_d1\_uncomp.avi  
📺 person01\_jogging\_d2\_uncomp.avi  
📺 person01\_jogging\_d3\_uncomp.avi  
📺 person01\_jogging\_d4\_uncomp.avi  
📺 person02\_jogging\_d1\_uncomp.avi  
📺 person02\_jogging\_d2\_uncomp.avi  
📺 person02\_jogging\_d3\_uncomp.avi  
📺 person02\_jogging\_d4\_uncomp.avi

...

📺 person25\_jogging\_d1\_uncomp.avi  
📺 person25\_jogging\_d2\_uncomp.avi  
📺 person25\_jogging\_d3\_uncomp.avi  
📺 person25\_jogging\_d4\_uncomp.avi

Figure 4.6 Directory Structure of “jogging” Activity

📁 running

📺 person01\_running\_d1\_uncomp.avi  
📺 person01\_running\_d2\_uncomp.avi  
📺 person01\_running\_d3\_uncomp.avi  
📺 person01\_running\_d4\_uncomp.avi  
📺 person02\_running\_d1\_uncomp.avi  
📺 person02\_running\_d2\_uncomp.avi  
📺 person02\_running\_d3\_uncomp.avi  
📺 person02\_running\_d4\_uncomp.avi

...

📺 person25\_running\_d1\_uncomp.avi  
📺 person25\_running\_d2\_uncomp.avi  
📺 person25\_running\_d3\_uncomp.avi  
📺 person25\_running\_d4\_uncomp.avi

Figure 4.7 Directory Structure of “running” Activity

📁 walking

📺 person01\_walking\_d1\_uncomp.avi  
📺 person01\_walking\_d2\_uncomp.avi  
📺 person01\_walking\_d3\_uncomp.avi  
📺 person01\_walking\_d4\_uncomp.avi  
📺 person02\_walking\_d1\_uncomp.avi  
📺 person02\_walking\_d2\_uncomp.avi  
📺 person02\_walking\_d3\_uncomp.avi  
📺 person02\_walking\_d4\_uncomp.avi

...

📺 person25\_walking\_d1\_uncomp.avi  
📺 person25\_walking\_d2\_uncomp.avi  
📺 person25\_walking\_d3\_uncomp.avi  
📺 person25\_walking\_d4\_uncomp.avi

Figure 4.8 Directory Structure of “walking” Activity

### 4.2.2 Building the Sequential Model

The sequential model that is used to perform the activity recognition consists of a complex architecture with several parameters. It consists of several layers that are combined together to form the overall model. The basic structure of the sequential model is given in Figure 4.9.

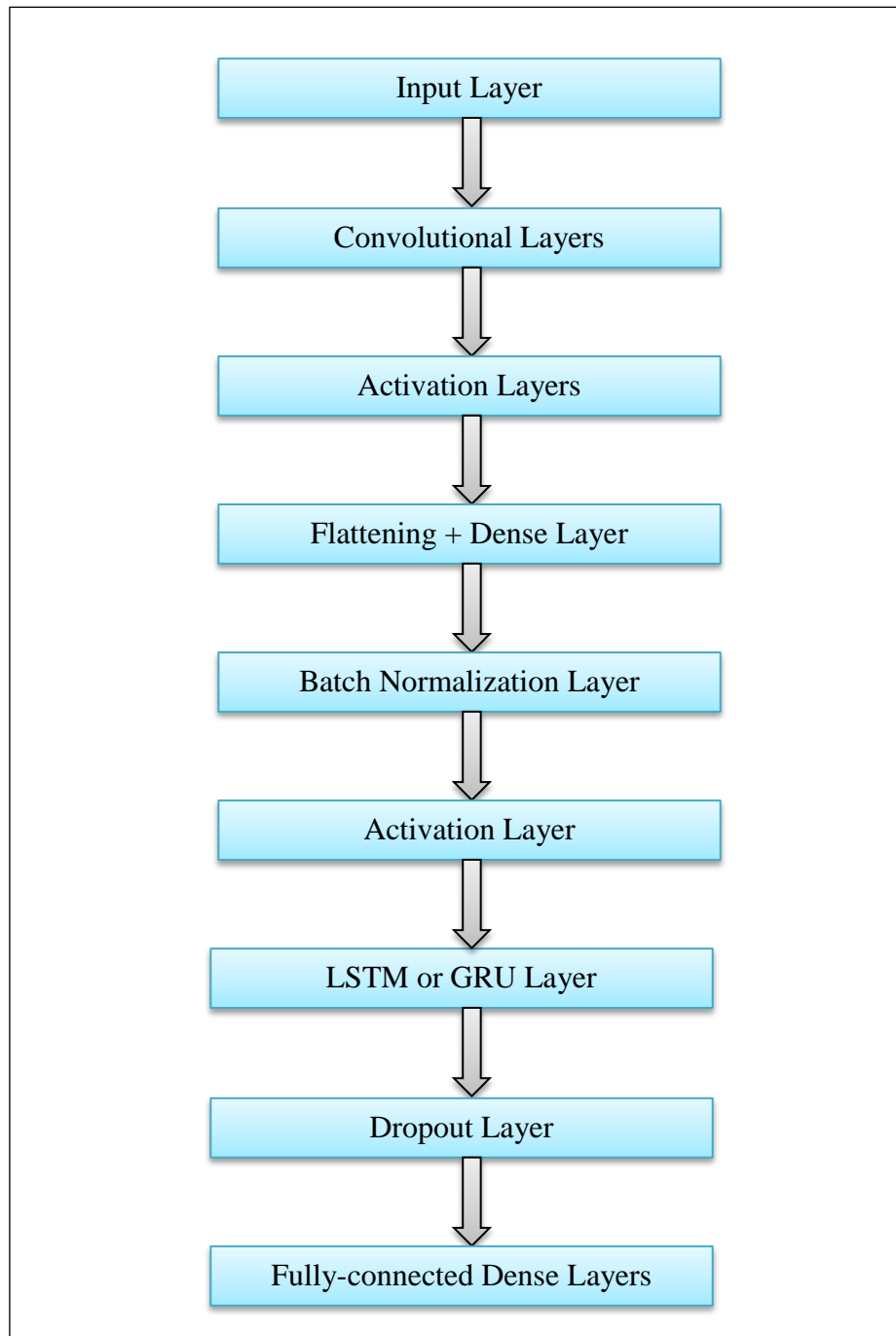


Figure 4.9 Structure of Sequential Model

## Chapter 5

### Detailed Design

The detailed design of the proposed system is given in terms of high level design and low level design. The following sections describe these designs thoroughly.

#### 5.1 High-Level Design

High-level design (HLD) explains the architecture that would be used for developing a software product. The architecture diagram provides an overview of an entire system, identifying the main components that would be developed for the product and their interfaces. The HLD uses possibly nontechnical to mildly technical terms that should be understandable to the administrators of the system.

The purpose of high-level design is given as follows:

- Preliminary design — in the preliminary stages of a software development, the need is to size the project and to identify those parts of the project that might be risky or time consuming.
- Design overview — as the project proceeds, the need is to provide an overview of how the various sub-systems and components of the system fit together.

In both cases the high-level design should be a complete view of the entire system, breaking it down into smaller parts that are more easily understood. To minimize the maintenance overhead as construction proceeds and the lower-level design is done, it is best that the high-level design is elaborated only to the degree needed to satisfy these needs.

##### 5.1.1 Design Consideration

The design consideration for the system involves the assumptions and dependencies of the system, mode of operation of the system and the subsequent user operations.

###### 5.1.1.1 Assumptions and Dependencies

Several assumptions and dependencies can be specified for human activity recognition being carried out in this project. They can be given as follows:

- The dataset is assumed to be complete and there are no missing videos and/or corrupted data.

- The dataset is also assumed to be untouched, i.e., no changes to be made to the dataset once it has been downloaded and stored locally.
- The required technology and software is assumed to have already been installed on the machine running the scripts required to perform human activity recognition.
- The project is dependent on the computer or machine having minimum specifications required to run the particular associated software.

#### **5.1.1.2 Mode of Operation of the System**

The proposed system in this project can be run only in the user mode. There is no interaction with the kernel specifically. All commands used to implement the functions in this project are made locally with respect to the OS and there is no need for any kernel interrupts.

When the system functions need to be invoked, subsequent calls are made and the necessary changes are made to the code that is being implemented. The system needs to be active throughout the running of the project, due to the fact that it is dependent on the completeness and availability of the dataset, as mentioned in the previous section.

#### **5.1.1.3 User Operations**

The interaction of the user with the system can be referred to as “user operations.” In this project, the user plays a very important role in administering the availability and completeness of the dataset. The user has to ensure that the dataset is complete and there are no corrupted/missing videos in the dataset. During implementation, the contribution of the user is vital in the form of providing input to the system in terms of training and test set parameters, and the number of epochs.

### **5.1.2 Data Flow Diagram**

A data flow diagram (DFD) is a graphical representation of the “flow” of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing.

A data flow diagram shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

The data flow diagram for the proposed system in this project can be given as shown in Figure 5.1. The input can be considered as the accumulation of the KTH Dataset, the functions include data pre-processing, conversion of the dataset videos into frames, building the model based on these frames, training the model with certain parameters, testing the model against certain parameters, and finally the tabulation of results and comparison of accuracies and log times of the different technologies/classifiers used.

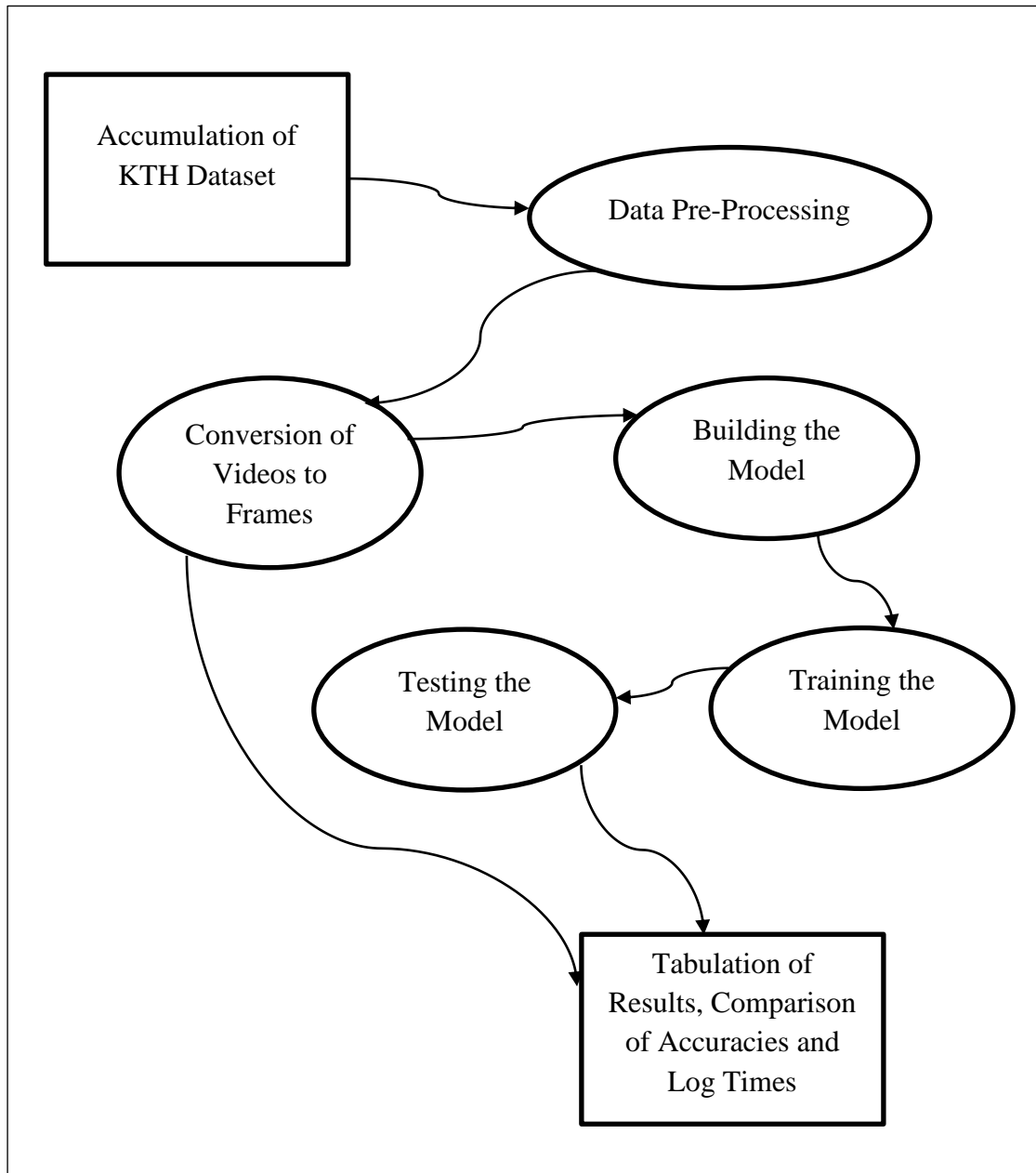


Figure 5.1 Data Flow Diagram of the System

## 5.2 Low-Level Design

Low-level design of the system includes the use case diagrams, sequence diagrams and class diagrams of all the components involved in the system.

### 5.2.1 Use case Diagram

A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The actors, usually individuals involved with the system defined according to their roles.

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

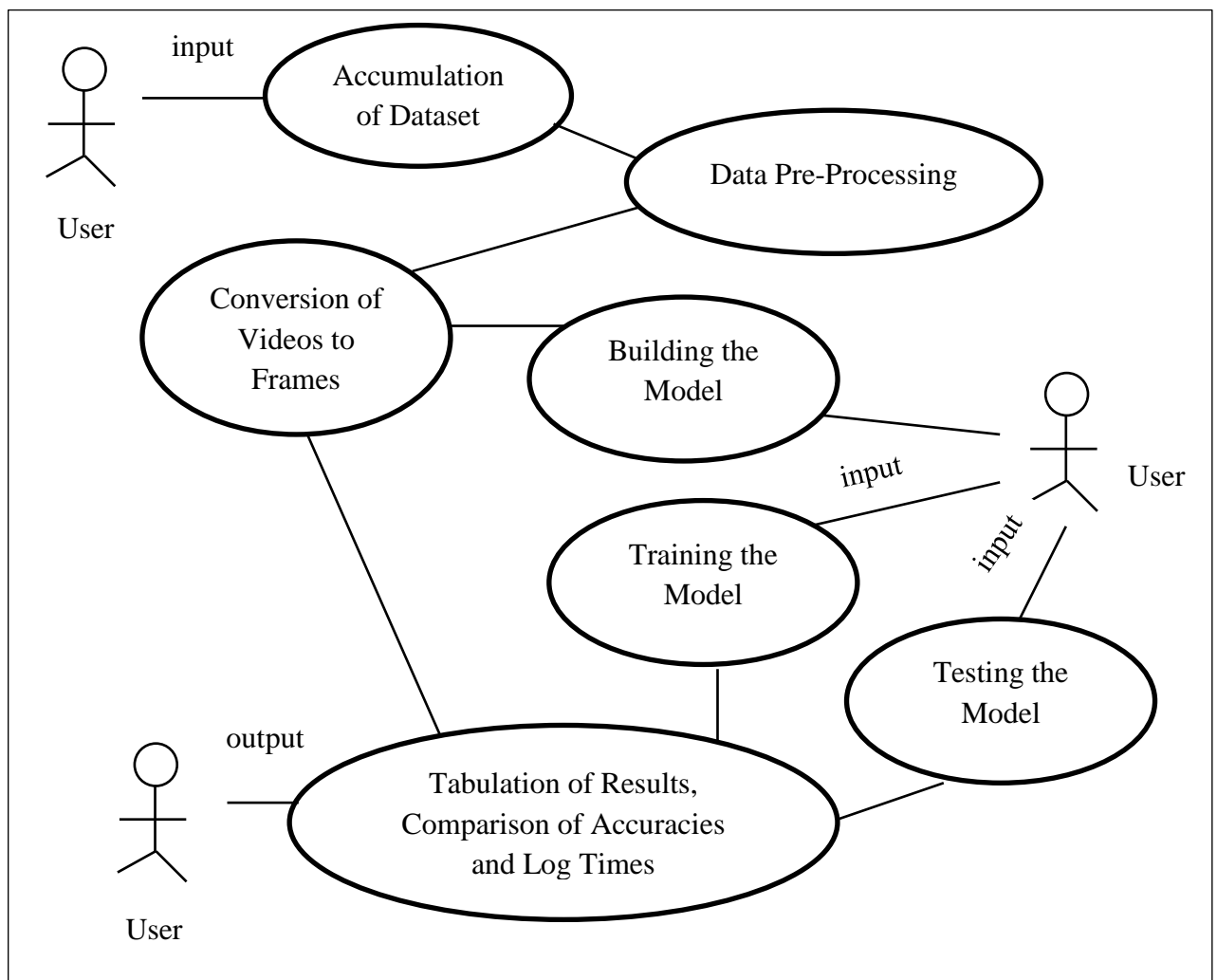


Figure 5.2 Use case Diagram of the System

In this project, the use case diagram shown in Figure 5.2 consists of several use cases in which the user is involved. The user initially obtains the KTH Dataset from available sources. After the system performs the initial conversion, the system builds the model and then asks for the input of training and testing parameters, including the number of epochs. The system then gives the output back to the user.

### 5.2.2 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

In this project, the interaction between the user and the system via the dataset and the model is depicted in the sequence diagram shown in Figure 5.3. Messages are passed between the user and the system, and subsequent responses/replies are received.

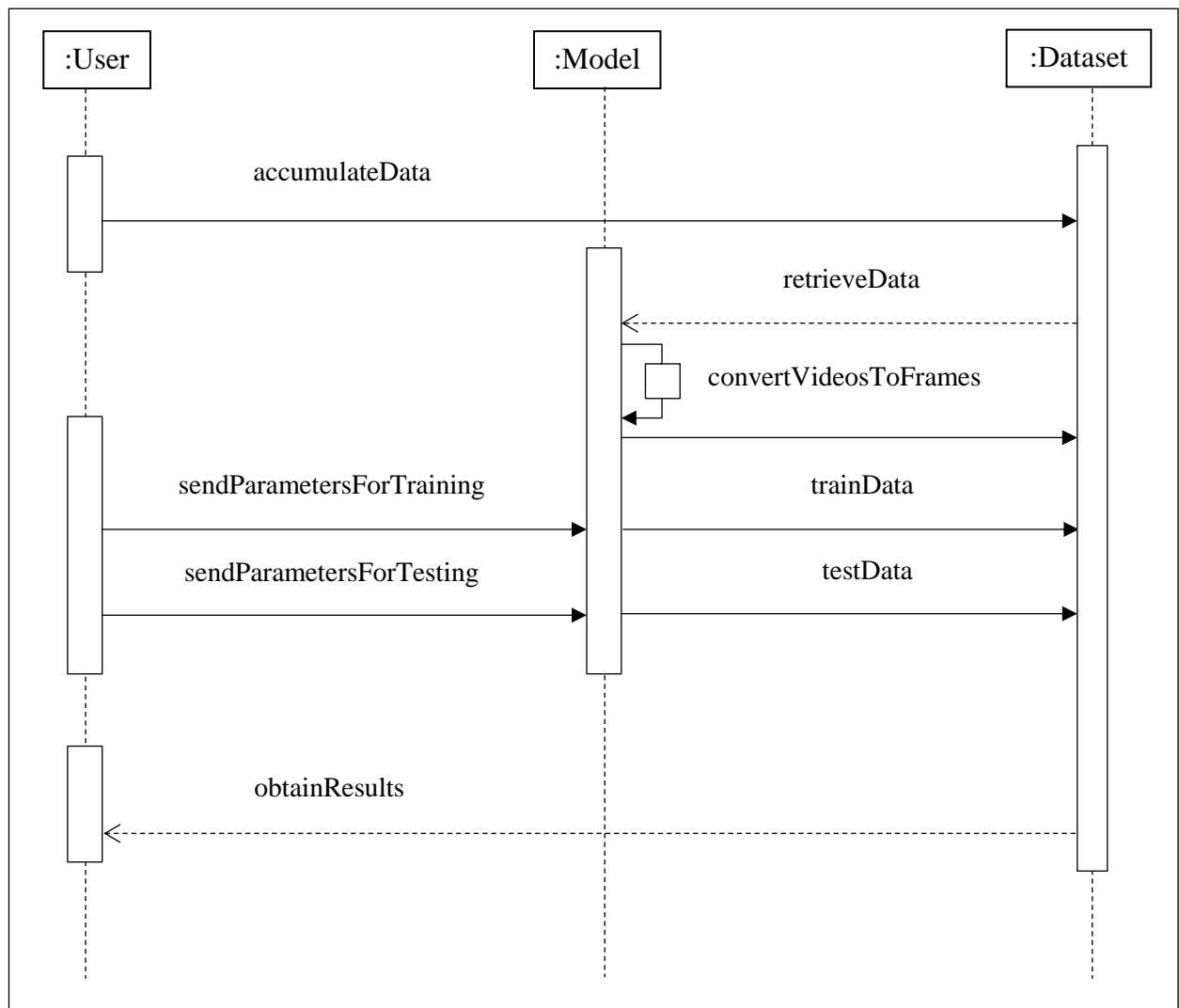


Figure 5.3 Sequence Diagram of the System

### 5.2.3 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.



In the class diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

In the design of a system, a number of classes are identified and grouped together in a class diagram that helps to determine the static relations between them. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

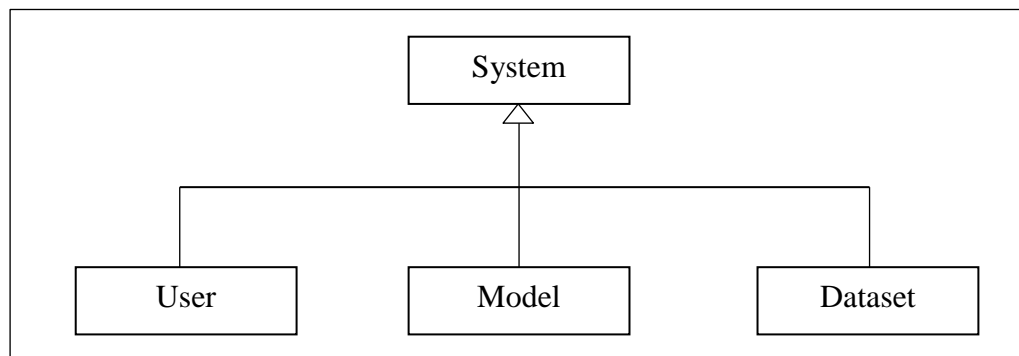


Figure 5.4 Class Diagram of the System

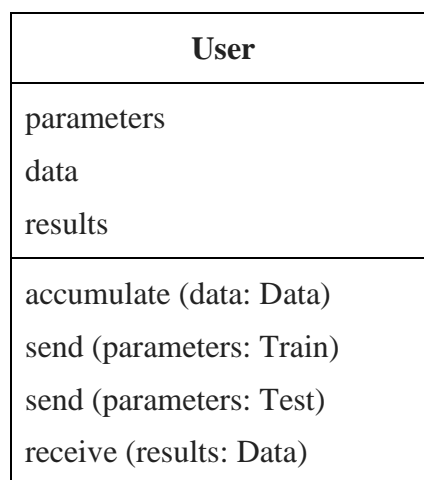


Figure 5.5 User Class

<b>Model</b>
parameters data model results
convert (data: Data) build (model: Model) receive (parameters: Train) train (model: Model) receive (parameters: Test) test (model: Model) evaluate (results: Data) send (results: Data)

Figure 5.6 Model Class

<b>Dataset</b>
data
store (data: Data)

Figure 5.7 Dataset Class

In this project, the main class is the System itself. This is shown with the class diagram for the entire project, in Figure 5.4. The classes being referred to are User, Model and Dataset. These are shown above in Figures 5.5, 5.6 and 5.7 respectively. Dependencies are formed between the classes in the system, such that there is relation between one another.

## Chapter 6

# Implementation Details

The implementation details of the project describe the procedure in which the project is implemented on the computer. This section includes the overview of the languages and frameworks used, the scripts that have to be run and the necessary software that have to be installed in order to perform Human Activity Recognition on the KTH Dataset. A brief description of the pseudo-codes/algorithms used is also included in this section.

### 6.1 Introduction

This project based on Human Activity Recognition uses Python as its primary language. The back-end is written using TensorFlow based on Keras API and the front-end is written using the Django framework. The complete guide to installing the necessary software and executing the project is given in the following sections.

### 6.2 Overview of System Implementation

The system implementation section consists of the usability and technical aspects of the project code. These aspects are necessary for anyone who is new to the system and needs to understand how to implement it based on the details.

#### 6.2.1 Usability Aspect

In software engineering, usability is the degree to which a software system can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use.

This project can be used by the general public with pre-requisites of the basics of machine learning and coding. The usage of some of the commands need to be referred to on the Internet, if need be, for further clarifications. Due to the fact that the proposed system is completely based on open-source resources, changes can be made to the existing code in order to satisfy any additional requirements, if any.

## **6.2.2 Technical Aspect**

The technical aspect of the implementation of this system lies in the accumulation and pre-processing of the KTH Dataset in accordance with the principles of Human Activity Recognition.

Further exploration in to the technical aspects leads us to the usage of various technologies in order to implement the data pre-processing techniques. A combined approach with respect to the sequential model built to carry out classification of the human activities is also considered to be a technical aspect of the system.

### **6.2.2.1 Dataset**

The KTH Dataset used in this project is open-source and can be used to carry out activity recognition based on the preferences of the user. The dataset consists of six different activities with four different scenarios for each activity for each person, so this gives a pretty wide variety of videos in order to perform Human Activity Recognition.

### **6.2.2.2 Sequential Model**

The sequential model consists of the combined approach of the different techniques used to conduct Human Activity Recognition, mainly CNN and LSTM/GRU. The model consists of a fixed input layer, followed by three layers of convolutions and their activations. This is further followed by a flattening layer. A dense layer along with its batch-normalized activation function ensures that the deep network learns, according to the inputs given. This is followed by either the LSTM or GRU layer, depending on which recurrent unit is chosen. Finally, we apply dropout to this model and finish off with fully connected dense layers.

The two different models used in this project are shown below in Figures 6.1 and 6.2 respectively.

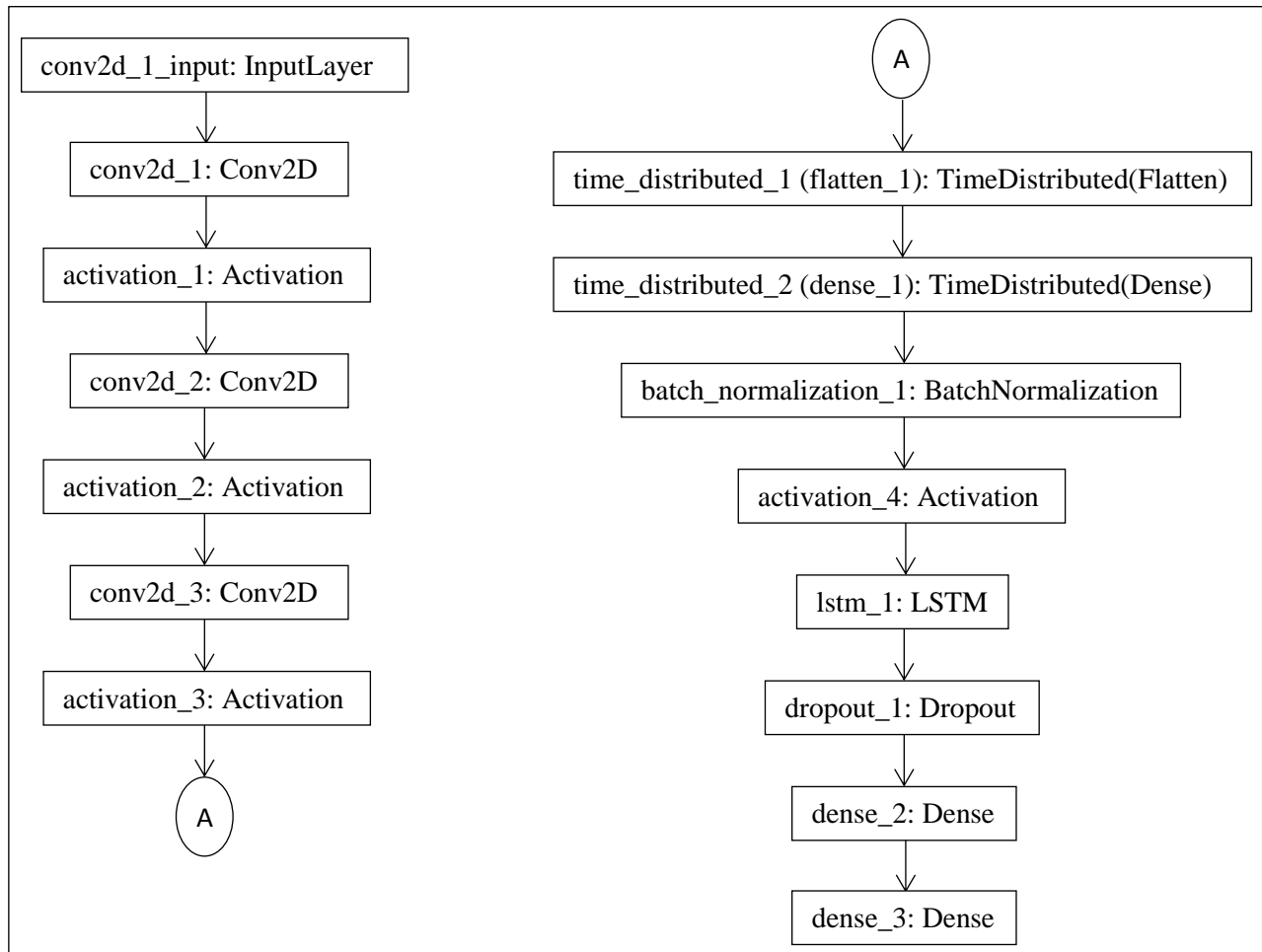


Figure 6.1 CNN-LSTM Sequential Model

Figure 6.1 represents the CNN-LSTM sequential model. Here, it is noticeable that the initial layer is an input layer to the model, which is used to take in input of the parameters. This is followed by the first 2-dimensional Convolutional Neural Network (CNN) layer along with its activation layer. Two more CNN layers along with their respective activation layers are defined further. The next layer in the sequential model is the flattening layer, which converts the tensor input from the previous layers to a compatible form before going to recurrent learning phase. A dense layer with a time-distributed setup is added next to ensure all neurons are connected to every other neuron in the previous layers. Batch normalization is applied next, to ensure that the system learns. The following activation layer performs the activation function based on the batch normalization. The Long Short-Term Memory (LSTM) layer defines the recurrent learning parameters, including the learning rate and the activation function for it. A small amount of dropout is applied to prevent overfitting. Finally, two layers of fully-connected dense neurons are used to conclude the sequential model.

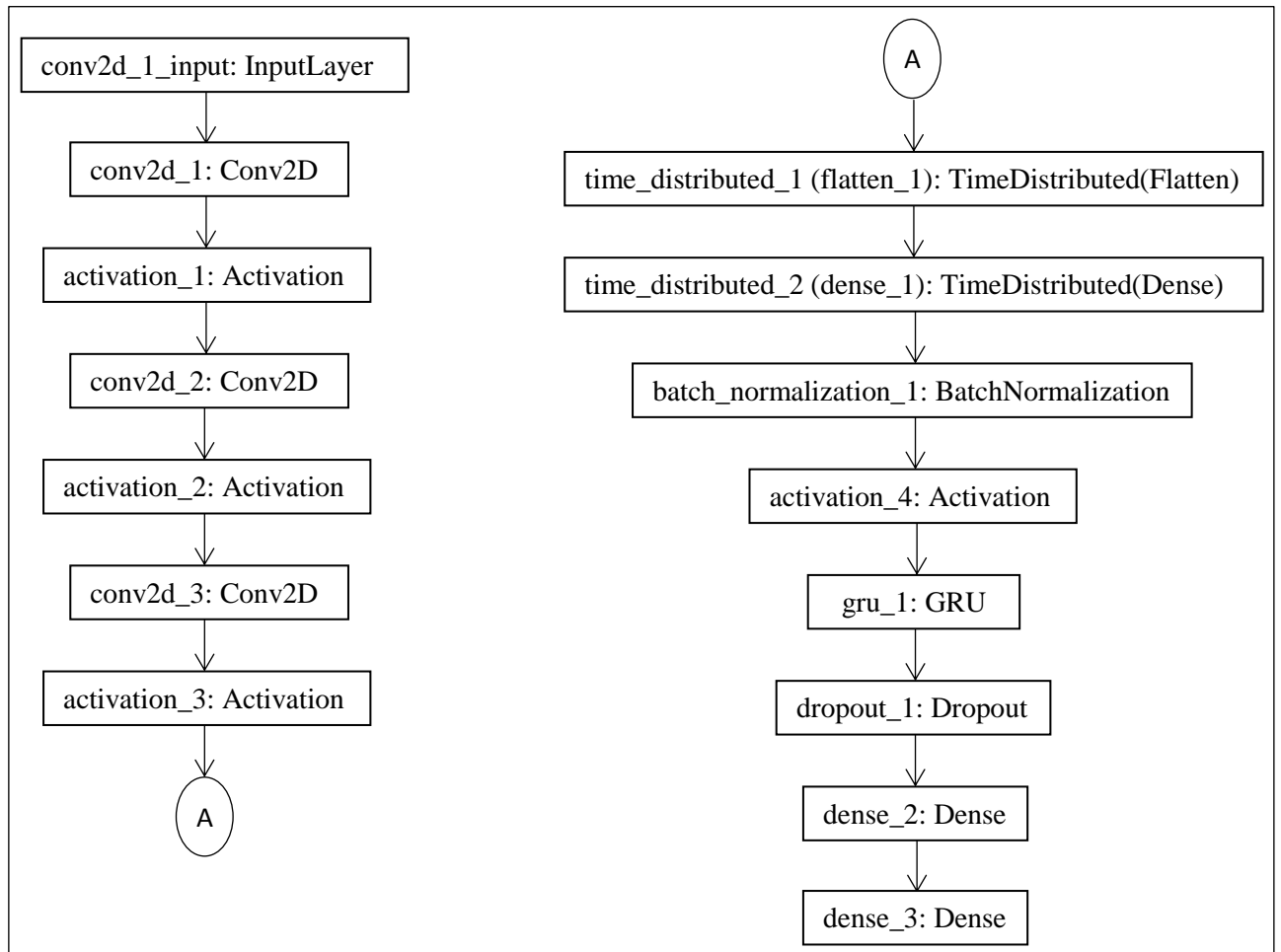


Figure 6.2 CNN-GRU Sequential Model

Figure 6.2 represents the CNN-GRU sequential model. Almost all the layers perform similar to the CNN-LSTM sequential model, except the Gated Recurrent Unit (GRU) layer. This layer in the model replaces the LSTM blocks with GRU blocks. A comparative analysis is made in order to determine which performs better for the entire dataset.

## 6.3 Implementation Support

This section includes the installation and setup of various technologies and software used to implement Human Activity Recognition on the KTH Dataset.

### 6.3.1 Installing Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that

emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

In this project, Python 3.6 is used as the basic programming language to type all the functions and import libraries necessary to perform human activity recognition. Once installed, the main executable is added to the system environment variable PATH for execution of Python scripts and configuration of interpreters.

### **6.3.2 Installing PyCharm Community Edition**

PyCharm is an Integrated Development Environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django.

The Community Edition of PyCharm is an open-source version of PyCharm, and is released under the Apache License. It is used in this project to efficiently type, compile, debug and run Python scripts in the back-end.

### **6.3.3 Installing TensorFlow**

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

TensorFlow provides excellent resources to perform machine learning, especially deep learning with the Keras API. The ability to work on deep learning on a CPU alone is a massive advantage when it comes to activity recognition, due to the fact that the dataset exists on the local hard disk drive (HDD).

### 6.3.4 Installing Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or MXNet. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIRO (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

Keras supports both convolutional and recurrent networks, as well as combinations of the two. In this project, the usage of combination of CNN and LSTM/GRU runs extremely well on Keras. A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that can be combined to create new models.

### 6.3.5 Installing NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a collection of high level mathematical functions to operate on these arrays. The core functionality of NumPy is its "ndarray", for  $n$ -dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

In this project, NumPy has been extensively used to store data as  $n$ -dimensional data structures, in order to receive, manipulate and send data across different functions.

### 6.3.6 Installing FFmpeg and OpenCV

FFmpeg is a free software project, the product of which is a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. At its core is the FFmpeg program itself, designed for command-line-based processing of video and audio files, widely used for format transcoding, basic editing (trimming and concatenation), video scaling, video post-production effects, and standards compliance (SMPTE, ITU).



OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real time computer vision. OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

In this project, both these technologies are used to perform the conversion of videos present in the KTH Dataset to frames. A brief comparative analysis is made between the two technologies based on the time taken to convert the videos to frames and also the size of the subsequent frames.

### **6.3.7 Installing Matplotlib, Graphviz and Pydot**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented application programming interface (API) for embedding plots into applications using general-purpose graphical user interface (GUI) toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural “pylab” interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB.

Graphviz (short for *Graph Visualization Software*) is a package of open-source tools initiated by AT&T Labs Research for drawing graphs specified in DOT language scripts. It also provides libraries for software applications to use the tools. Graphviz is free software licensed under the Eclipse Public License.

Pydot is a Python interface to GraphViz and the DOT language. This package includes an interface to GraphViz, with classes to represent graphs and dump them in the DOT language, and a parser from DOT.

In this project, a combination of all these three technologies is used to display the final output of the activity recognition, that is, the confusion matrix based on the predicted and true labels of the activities.

### 6.3.8 Installing h5py

The h5py package is a Pythonic interface to the HDF5 binary data format. It lets the user store huge amounts of numerical data, and easily manipulate that data from NumPy. Both high-level and low-level access to HDF5 (Hierarchical Data Format 5) abstractions are provided in the h5py package.

In this project, h5py is used to store the final architecture and the parameters in a .h5 file. Along with h5py, the model parameters are stored in a JSON (JavaScript Object Notation) string for clarity.

### 6.3.9 Installing Django

Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. It is maintained by the Django Software Foundation (DSF). Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of "don't repeat yourself". Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

It is used as the main framework for the front-end in this project. Pages are made in order to give a brief description about the concept of activity recognition and the KTH dataset. The input of training and test parameters along with the number of epochs is also designed as a page.

## 6.4 Pseudocodes

This section gives snapshots of some of the pseudocodes used in this project, with respect to the back-end and the front-end.

The main pseudocodes are shown in Figures 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 respectively. These codes are all typed in the PyCharm IDE and are compiled and run using the Python interpreter that is installed in the system and after the PATH system environment variable has been configured.

The technologies and software mentioned in the previous section need to be installed beforehand in order to obtain the precise output. The times taken to execute these codes may differ from system to system, based on the configuration of the respective system.

```
log the start time
for x from 1 to number of class labels:
    specify class folder paths
    create folders and subfolders using mkdir
open indices file in read mode
for j from 1 to number of persons:
    read each line from indices file
    for k from 1 to number of scenarios:
        make folder to store frames
        if line in file matches subfolder name:
            convert video to frames using ffmpeg command
            make segments
            for p from 1 to number of segments:
                create segment folders
                for q between range of segment frames:
                    move frames to segments
        else:
            print "Invalid index"
close indices file
log the finish time
```

Figure 6.3 Pseudocode to Convert Videos to Frames using FFmpeg

Figure 6.3 shows the pseudocode that is used to convert the videos present in the KTH Dataset to frames. This is done by individually fetching each video using a continuous branching of loops and then running the FFmpeg in-built command that converts each video into frames based on the number of frames it is able to generate.

```
log the start time
for x from 1 to number of class labels:
    specify class folder paths
    create folders and subfolders using mkdir
open indices file in read mode
for j from 1 to number of persons:
    read each line from indices file
    for k from 1 to number of scenarios:
        make folder to store frames
        if line in file matches subfolder name:
            capture the video file
            count number of frames
            convert video to frames using cv2 cap methods
            make segments
            for p from 1 to number of segments:
                create segment folders
                for q between range of segment frames:
                    move frames to segments
        else:
            print "Invalid index"
close indices file
log the finish time
```

Figure 6.4 Pseudocode to Convert Videos to Frames using OpenCV

Figure 6.4 shows the pseudocode that is used to convert videos present in the KTH Dataset to frames using the OpenCV technology. This is done by counting the number of frames that each video has, using an in-built function and then, the conversion is done using another in-built OpenCV function that uses encoders and decoders.

```
for i from 1 to number of class labels:
    specify class folder
    for j from start person index to finish person index:
        for k from 1 to number of scenarios:
            specify recording folder
            for m from 1 to number of segments:
                specify segment folder
                get list of files
                sample the frames
                create list for each segment
                pre-processing methods
create one-hot encoded vectors
make numpy array for particular person data
return numpy array and one-hot vector
```

Figure 6.5 Pseudocode to Load Data

Figure 6.5 shows the pseudocode to load data for each person before the model is built. This is done by specifying the path to the frames present in the segmented folders and running a loop to collect all of them. The subsequent data is stored in NumPy arrays and 16 random samples are generated for each person, for each activity that they perform. OpenCV is used to read the file paths. Finally, the obtained array is one-hot encoded to obtain a multi-dimensional array based on the class labels, and this array is returned as the training and/or testing data parameters.

```
number of rows in image = 120
number of columns in image = 160
number of classes = 6
log the start time
specify sequential model being used
add first convolution layer with input shape of rows and columns
add its activation function
add two more convolution layers and their activation functions
flatten the input and prepare for recurrent learning
add a single dense layer
batch normalization of layers
add LSTM/GRU layer
add a small dropout
add two fully-connected dense layers
compile with loss function and optimizer
print model summary
plot model to an image file
log the end time
```

Figure 6.6 Pseudocode to Build Model and Plot Model Architecture

Figure 6.6 shows the pseudocode that is used to build the sequential model using Keras commands. The Keras API provides in-built libraries that are used to add layers to the sequential model individually.

First, the three two-dimensional convolutional layers are defined one after the other, along with their corresponding activation functions. A Rectified Linear Unit (ReLU) is used as the activation function for these layers. A time-distributed flattening layer follows these layers and it is used to flatten the extra dimension of time that is present in the videos. This is followed by a time-distributed dense layer. A layer of batch normalization along with its activation function follows this dense layer, which ensures that the system learns.

This is followed by the LSTM/GRU layer, that has a hyperbolic tangent function as its activation function and which consists of 80 recurrent units. A dropout layer is added after this layer to prevent over-fitting of data. Finally, a couple of dense layers are used to fully connect the model

and it is then passed for compilation using a loss function of categorical cross-entropy and the RMSprop optimizer.

A summary of the model is printed for the user to understand the layers involved and the model is saved and plotted as an image file in the local hard disk drive.

```
batch size = 64
number of epochs = 30
log the start load time
load data for persons within given training range
log the end load time
log the start train time
train using model.fit()
clean up the memory, for testing
log the end train time
log the start test time
load data for persons within given testing range
get predictions using model.predict()
calculate accuracy
plot confusion matrix
log the end test time
output log times, accuracy and confusion matrix
save confusion matrix in an image file
save model parameters in a JSON string
save model architecture and weights in a HDF5 file
```

Figure 6.7 Pseudocode to Train & Test Model, Calculate & Plot Confusion Matrix, Calculate Accuracy and Save Parameters & Model Architecture

Figure 6.7 shows the pseudocode that is written to train & test the model, calculate & plot the confusion matrix, calculate the validation accuracy and also save the parameters and the model architecture.

Firstly, the indexes of the persons needed to train and test the model are input. This is then fit into the model using the fit function provided by the Keras API. The predictions are then made using the predict function that is followed by the calculation of the accuracy and confusion matrix. The

confusion matrix is then plotted and saved in an image file by using the functions provided by Matplotlib, Graphviz and pydot. The respective model is saved in a JSON string on the local hard disk and the architecture and its weights are saved in a HDF5 file using the h5py package functions.

```
import render and HTTP libraries
define start page:
    provide link to next HTML page
define page2:
    display information
    provide link to next HTML page
define page3:
    display information
    provide link to next HTML page
define input page:
    take training, testing parameters, number of epochs as input
    pass these values to a text file
    provide link to output HTML page
define writing to file:
    open text file in write mode
    write parameters line by line
    call main program script using system call
    return parameters to output page
```

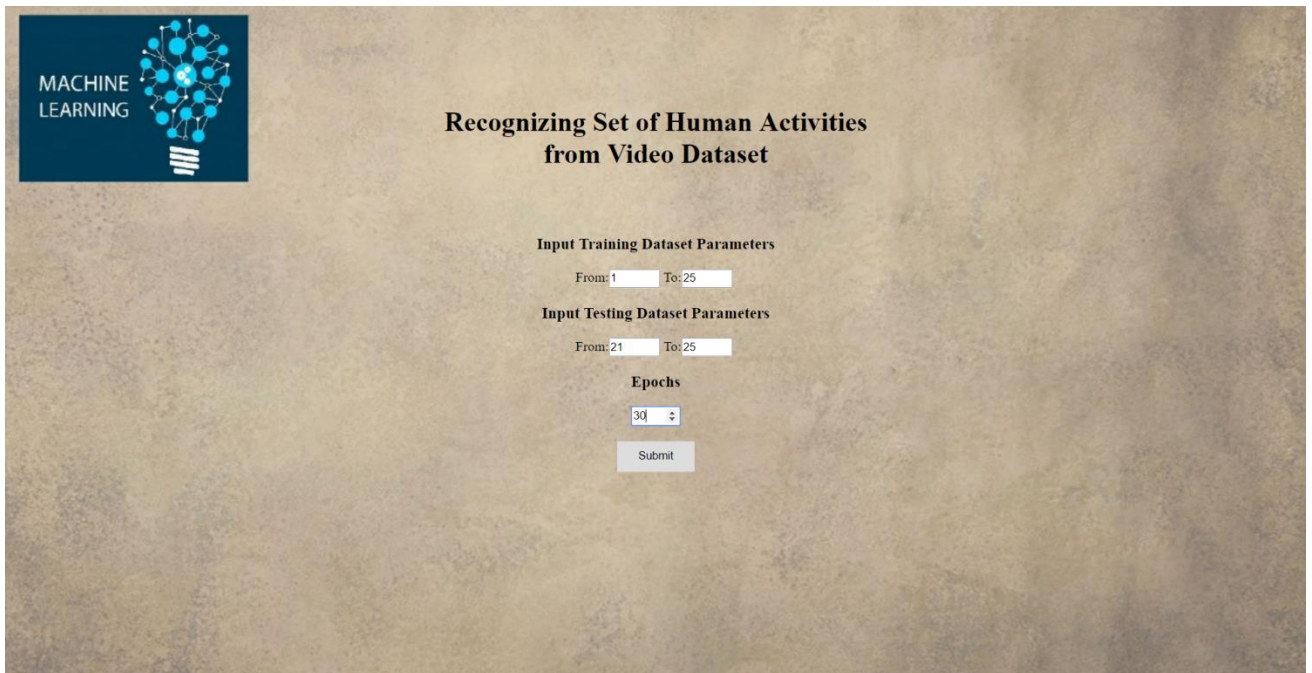
Figure 6.8 Pseudocode to Display Front-End Views

Figure 6.8 shows the necessary views that have to be defined in Django for the efficient functioning of the front-end. This involves the in-built render functions and the GET and POST methods.



## 6.5 Front End Design

This section describes the front-end that takes in the input of training and testing set parameters, in terms of person indexes ranging from 1 to 25. It also takes in the input of number of epochs that are input to the back-end for training, ranging from 10 to 200. Figure 6.9 shows the front-end that is displayed to the user.



The screenshot displays a web application interface with a dark blue header on the left containing the text "MACHINE LEARNING" and a network diagram icon. The main content area has a light beige background and is titled "Recognizing Set of Human Activities from Video Dataset". Below the title, there are three input sections: "Input Training Dataset Parameters" with "From: 1" and "To: 25" text boxes; "Input Testing Dataset Parameters" with "From: 21" and "To: 25" text boxes; and "Epochs" with a dropdown menu showing "30". A "Submit" button is located at the bottom of these input fields.

Figure 6.9 Front-End to Receive Inputs

# Testing

Testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.

### 7.1 Introduction

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- Meets the requirements that guided its design and development.
- Responds correctly to all kinds of inputs.
- Performs its functions within an acceptable time.
- Is sufficiently usable.
- Can be installed and run in its intended environments.
- Achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones. Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs.

## 7.2 Levels of Testing

There are a total of five levels of testing, namely unit testing, integration testing, system testing, validation testing and user acceptance testing.

### 7.2.1 Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

In this project work, unit testing can be implemented under two evaluations, that is, user input and error handling. The test cases for unit testing are depicted in Tables 7.1 and 7.2, as shown below.

#### 7.2.1.1 User Input

The values that the user inputs as the training and testing parameters, and the number of epochs, are considered to be user input. These inputs are tested with various test cases, with respect to different person numbers and different number of epochs.

#### 7.2.1.2 Error Handling

Error handling is done in this project by checking the range of the user inputs, that is, the training and testing parameters, and the number of epochs. Whenever a user inputs a value that is not within the range of the training or testing parameters (1 to 25), an error message pops up in the front-end, informing the user that an invalid input is given, and the subsequent code has stopped executing. Whenever a user inputs a value of epochs that is not within the range (10 to 100), then an error message informs the user that the number of epochs entered is not a valid number.

Table 7.1 Test Cases for Unit Testing of Frame Conversion

No.	Test Case	Expected Output	Actual Output	Status
1	Dataset not present in the target folder	File not found	File not found	Pass
2	Dataset corrupted or incomplete	File not found	File not found	Pass
3	Invalid data in the indices file	Print "Invalid Index"	Print "Invalid Index"	Pass
4	Videos already converted to frames	File exists	File exists	Pass

Table 7.2 Test Cases for Unit Testing of Sequential Model

No.	Test Case	Expected Output	Actual Output	Status
1	Recordings not present in the segment folder	File not found	File not found	Pass
2	Input invalid index for training parameter (start index < 1 and finish index > 25)	Invalid range	Invalid range	Pass
3	Input invalid index for testing parameter (start index < 1 and finish index > 25)	Invalid range	Invalid range	Pass
4	Input invalid value for number of epochs (epochs < 10 and epochs > 200)	Invalid range	Invalid range	Pass

### 7.2.2 Integration Testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

In this project work, the individual modules built to convert videos into frames are tested as a group. These are then combined into larger groups of code for both technologies and testing is done. The test cases for integration testing are given in Table 7.3, as shown below.

Table 7.3 Test Cases for Integration Testing

No.	Test Case	Expected Output	Actual Output	Status
1	Accumulated dataset sorted into segmented folders	Satisfactory data pre-processing	Satisfactory data pre-processing	Pass
2	Segments made according to sequences list	Satisfactory data pre-processing	Satisfactory data pre-processing	Pass
3	Log the time of building the sequential model	Satisfactory time observed	Satisfactory time observed	Pass
4	Log the time of training the model	Satisfactory time observed	Satisfactory time observed	Pass
5	Log the time of testing the model	Satisfactory time observed	Satisfactory time observed	Pass
6	Calculate accuracy and confusion matrix	Satisfactory results obtained	Satisfactory results obtained	Pass

### 7.2.3 System Testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

In this project work, system testing is done on the entire model that is built based on the input parameters, namely training set parameters, testing set parameters and number of epochs. The test cases are shown in Table 7.4 below.

Table 7.4 Test Cases for System Testing

No.	Input			Expected Output	Actual Output	Status
	Training	Testing	Epochs			
1	1 – 10	4 – 4	30	High accuracy	High accuracy	Pass
2	1 – 10	4 – 6	30	High accuracy	High accuracy	Pass
3	1 – 20	21 – 25	30	Moderate accuracy	Low accuracy	Fail
4	1 – 25	21 – 25	30	Highest accuracy	Highest accuracy	Pass
5	1 – 20	21 – 25	50	Moderate accuracy	Moderate accuracy	Pass
6	1 – 20	21 – 25	100	High accuracy	Moderate accuracy	Fail
7	1 – 20	21 – 25	200	High accuracy	Moderate accuracy	Fail
8	1 – 20	19 – 25	100	Moderate accuracy	Moderate accuracy	Pass
9	1 – 20	15 – 25	100	High accuracy	High accuracy	Pass

### 7.2.4 Validation Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements. Validation Testing ensures that the product actually meets the client's needs.

It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment. Validation can be considered as the combination of four different activities, namely unit testing, integration testing, system testing and acceptance testing.

### 7.2.5 User Acceptance Testing

User acceptance testing (UAT) consists of a process of verifying that a solution works for the user. Software vendors often refer to this as "Beta testing". This testing should be undertaken by a subject-matter expert (SME), preferably the owner or client of the solution under test, and provide a summary of the findings for confirmation to proceed after trial or review.

In software development, UAT as one of the final stages of a project often occurs before a client or customer accepts the new system. Users of the system perform tests in line with what would occur in real-life scenarios. In this project work, the most acceptable values for the input parameters are considered for user acceptance testing. The results are noted down and subsequent alterations are made to the system.

## Chapter 8

### Discussion of Results

The results observed in this project are based on accuracy and confusion matrix of the class labels that are classified. The entire work has been carried out in the PyCharm IDE and subsequent results noted down using the command console. Variables are defined in the code that log the time taken to complete the respective actions such as loading the data, training the data and testing the data.

#### 8.1 Log Times of FFmpeg versus OpenCV

The conversion of videos to frames is the first major part of this project. This is done by the use of two different technologies – FFmpeg and OpenCV. The time taken to complete the conversion of videos in the KTH dataset to frames is considered and compared. The results are shown in Table 8.1 below.

Table 8.1 Log Times of FFmpeg vs. OpenCV

<b>Trial No.</b>	<b>FFmpeg Log Time (in seconds)</b>	<b>OpenCV Log Time (in seconds)</b>
1	1695.91	1985.83
2	1535.98	2200.98
3	1700.03	1998.87
4	1677.23	2001.22
5	1630.09	2056.76
6	1580.75	2026.11
7	1712.26	1990.54

It is observed from the above table that FFmpeg takes lesser time to convert the videos to frames. This is possible because of the better encoding library present in FFmpeg. It is also observed that the size of the frames that are generated by FFmpeg are far less than those of OpenCV, with which we can conclude that, for smaller datasets such as the KTH Dataset, FFmpeg is a better option to convert videos to frames.

## 8.2 Log Times and Accuracies of LSTM and GRU

LSTM and GRU are two techniques that are used to combine the convolutional neural networks in order to obtain better results for the classification of activities in the KTH Dataset. The log times and respective accuracies are compared between these two techniques. The evaluation metrics – accuracy and confusion matrix are defined first.

### 8.2.1 Performance Measures

The calculation of accuracy and plotting of the confusion matrix require certain concepts that have to be understood. These are given as follows:

- True positives (TP) are test cases classified rightly as positive cases.
- True negatives (TN) are test cases classified rightly as negative cases.
- False positives (FP) are test cases classified as positive but are negative cases.
- False negatives (FN) are test cases classified as negative but are positive cases.

#### 8.2.1.1 Accuracy

Accuracy has two definitions:

- More commonly, it is a description of systematic errors, a measure of statistical bias; as these cause a difference between a result and a "true" value, ISO calls this trueness.
- Alternatively, the International Standards Organization (ISO) defines accuracy as describing a combination of both types of observational error above (random and systematic), so high accuracy requires both high precision and high trueness.

In simplest terms, given a set of data points from a series of measurements, the set can be measured, while the set can be said to be accurate if the values are close to the true value of the quantity being measured. The two concepts are independent of each other, so a particular set of data can be said to be either accurate, or precise, or both, or neither.

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



### 8.2.1.2 Confusion Matrix

Confusion matrix contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix.

		Predicted Label	
		Positive	Negative
True Label	Positive	TP (Number of TPs)	FN (Number of FNs)
	Negative	FP (Number of FPs)	TN (Number of TNs)

Figure 8.1 Confusion Matrix Definition

### 8.2.2 LSTM versus GRU

In this section, the comparison of LSTM and GRU recurrent building units are made in terms of their parameters – model build times, data load times, training and testing times, and their validation accuracies.

Long short-term memory (LSTM) units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM.

Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feed forward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell.

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks. Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory.

However, GRUs have been shown to exhibit better performance on smaller datasets. They have fewer parameters than LSTM, as they lack an output gate. They combine the forget and input

gates into a single “update gate.” They also merge the cell state and hidden state. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

The tables shown below compare these two units in terms of various parameters and subsequent observations are made.

Table 8.2 Comparison of Build Model Times of LSTM and GRU

<b>Trial No.</b>	<b>Epochs</b>	<b>Train Set (Person Indexes)</b>	<b>Test Set (Person Indexes)</b>	<b>Test Samples</b>	<b>Build Model Time of LSTM (in seconds)</b>	<b>Build Model Time of GRU (in seconds)</b>
1	30	1 - 10	4 – 4	96	0.52	0.47
2	30	1 - 10	4 – 6	288	0.51	0.47
3	30	1 - 20	21 – 25	480	0.50	0.46
4	30	1 - 25	21 – 25	480	0.51	0.50
5	50	1 - 20	21 – 25	480	0.54	0.49
6	100	1 - 20	21 – 25	480	0.51	0.47
7	200	1 - 20	21 – 25	480	0.54	0.47
8	100	1 - 20	19 – 25	672	0.50	0.45
9	100	1 - 20	15 – 25	1056	0.58	0.48

From Table 8.2, it can be observed that the build model times of GRU are lesser than those of LSTM for the respective trials. This is due to the fact that there is a lack of an output gate in the GRU, hence there are significantly lesser number of parameters in GRU to build.

Table 8.3 Comparison of Load Data Times of LSTM and GRU

<b>Trial No.</b>	<b>Epochs</b>	<b>Train Set (Person Indexes)</b>	<b>Test Set (Person Indexes)</b>	<b>Test Samples</b>	<b>Load Data Time of LSTM (in seconds)</b>	<b>Load Data Time of GRU (in seconds)</b>
1	30	1 - 10	4 – 4	96	99.19	103.18
2	30	1 - 10	4 – 6	288	98.95	98.49
3	30	1 - 20	21 – 25	480	393.63	365.53
4	30	1 - 25	21 – 25	480	561.43	563.02
5	50	1 - 20	21 – 25	480	365.82	365.72
6	100	1 - 20	21 – 25	480	368.35	361.44
7	200	1 - 20	21 – 25	480	365.47	364.67
8	100	1 - 20	19 – 25	672	365.85	362.84
9	100	1 - 20	15 – 25	1056	365.36	365.09

From Table 8.3, it is evident that there is not a lot of difference between the load data times of LSTM and GRU. This is due to the fact that loading the dataset is not dependent on the parameters of the model.

Table 8.4 Comparison of Training Times of LSTM and GRU

<b>Trial No.</b>	<b>Epochs</b>	<b>Train Set (Person Indexes)</b>	<b>Test Set (Person Indexes)</b>	<b>Test Samples</b>	<b>Training Time of LSTM (in seconds)</b>	<b>Training Time of GRU (in seconds)</b>
1	30	1 - 10	4 – 4	96	62.27	61.39
2	30	1 - 10	4 – 6	288	62.65	61.83
3	30	1 - 20	21 – 25	480	122.59	121.36
4	30	1 - 25	21 – 25	480	153.33	151.32
5	50	1 - 20	21 – 25	480	202.81	201.16
6	100	1 - 20	21 – 25	480	402.55	399.61
7	200	1 - 20	21 – 25	480	809.72	796.59
8	100	1 - 20	19 – 25	672	406.84	397.71
9	100	1 - 20	15 – 25	1056	406.46	398.72

Table 8.5 Comparison of Testing Times of LSTM and GRU

<b>Trial No.</b>	<b>Epochs</b>	<b>Train Set (Person Indexes)</b>	<b>Test Set (Person Indexes)</b>	<b>Test Samples</b>	<b>Testing Time of LSTM (in seconds)</b>	<b>Testing Time of GRU (in seconds)</b>
1	30	1 - 10	4 – 4	96	2.57	2.61
2	30	1 - 10	4 – 6	288	12.33	12.31
3	30	1 - 20	21 – 25	480	43.87	28.93
4	30	1 - 25	21 – 25	480	28.96	28.67
5	50	1 - 20	21 – 25	480	28.69	28.75
6	100	1 - 20	21 – 25	480	28.89	28.45
7	200	1 - 20	21 – 25	480	28.83	29.26
8	100	1 - 20	19 – 25	672	52.21	52.27
9	100	1 - 20	15 – 25	1056	118.42	118.75

From Tables 8.4 and 8.5, it can be observed that the training times of LSTM unit are far higher than those of GRU, because of the higher number of training parameters in the model. It can also be observed that the testing times of both the units are similar; this is due to the fact that the testing function is not dependent on the model parameters.

Table 8.6 Comparison of Accuracies of LSTM and GRU

<b>Trial No.</b>	<b>Epochs</b>	<b>Train Set (Person Indexes)</b>	<b>Test Set (Person Indexes)</b>	<b>Test Samples</b>	<b>Accuracy of LSTM (in %)</b>	<b>Accuracy of GRU (in %)</b>
1	30	1 - 10	4 – 4	96	93.75	96.88
2	30	1 - 10	4 – 6	288	82.29	90.625
3	30	1 - 20	21 – 25	480	43.75	47.5
4	30	1 - 25	21 – 25	480	98.125	93.75
5	50	1 - 20	21 – 25	480	50.625	43.54
6	100	1 - 20	21 – 25	480	53.75	50.21
7	200	1 - 20	21 – 25	480	48.96	50.625
8	100	1 - 20	19 – 25	672	63.98	60.71
9	100	1 - 20	15 – 25	1056	78.5	75.85

Table 8.6 shows the accuracies of LSTM and GRU with respect to changes in number of parameters, like epochs, training and test set and test samples. It can be observed that for smaller sets of data, GRU performs better than LSTM. This is because of the absence of an output gate, which increases the effectiveness with which classification is done.

Snapshots of the highest accuracies obtained with LSTM and GRU when the entire data set is taken into consideration, are displayed below, in Figures 8.2 and 8.3 respectively. Figures 8.4 and 8.5 depict sample confusion matrix outputs for LSTM and GRU respectively.

```
Number of epochs trained for: 30
Total no. of testing samples used: 480
Validation accuracy: 98.125 %
Confusion matrix:
['boxing', 'handclapping', 'handwaving', 'jogging', 'running', 'walking']
[[79.  0.  0.  0.  0.  0.]
 [ 1. 80.  1.  0.  0.  0.]
 [ 0.  0. 79.  0.  0.  0.]
 [ 0.  0.  0. 76.  0.  0.]
 [ 0.  0.  0.  2. 77.  0.]
 [ 0.  0.  0.  2.  3. 80.]]
Time taken to build the model: 0.5153706073760986
Time taken to load the data: 561.4349572658539
Time taken to train the samples: 153.32672333717346
Time taken to test the samples: 28.960010290145874
```

Figure 8.2 Snapshot of Highest Accuracy Obtained with LSTM for the Entire Dataset

```
Number of epochs trained for: 30
Total no. of testing samples used: 480
Validation accuracy: 93.75 %
Confusion matrix:
['boxing', 'handclapping', 'handwaving', 'jogging', 'running', 'walking']
[[80.  0.  0.  0.  0.  0.]
 [ 0. 80.  0.  0.  0.  0.]
 [ 0.  0. 80.  0.  0.  0.]
 [ 0.  0.  0. 77.  3. 10.]
 [ 0.  0.  0.  2. 76. 13.]
 [ 0.  0.  0.  1.  1. 57.]]
Time taken to build the model: 0.501319169998169
Time taken to load the data: 563.0171639919281
Time taken to train the samples: 151.3223934173584
Time taken to test the samples: 28.67023992538452
```

Figure 8.3 Snapshot of Highest Accuracy Obtained with GRU for the Entire Dataset

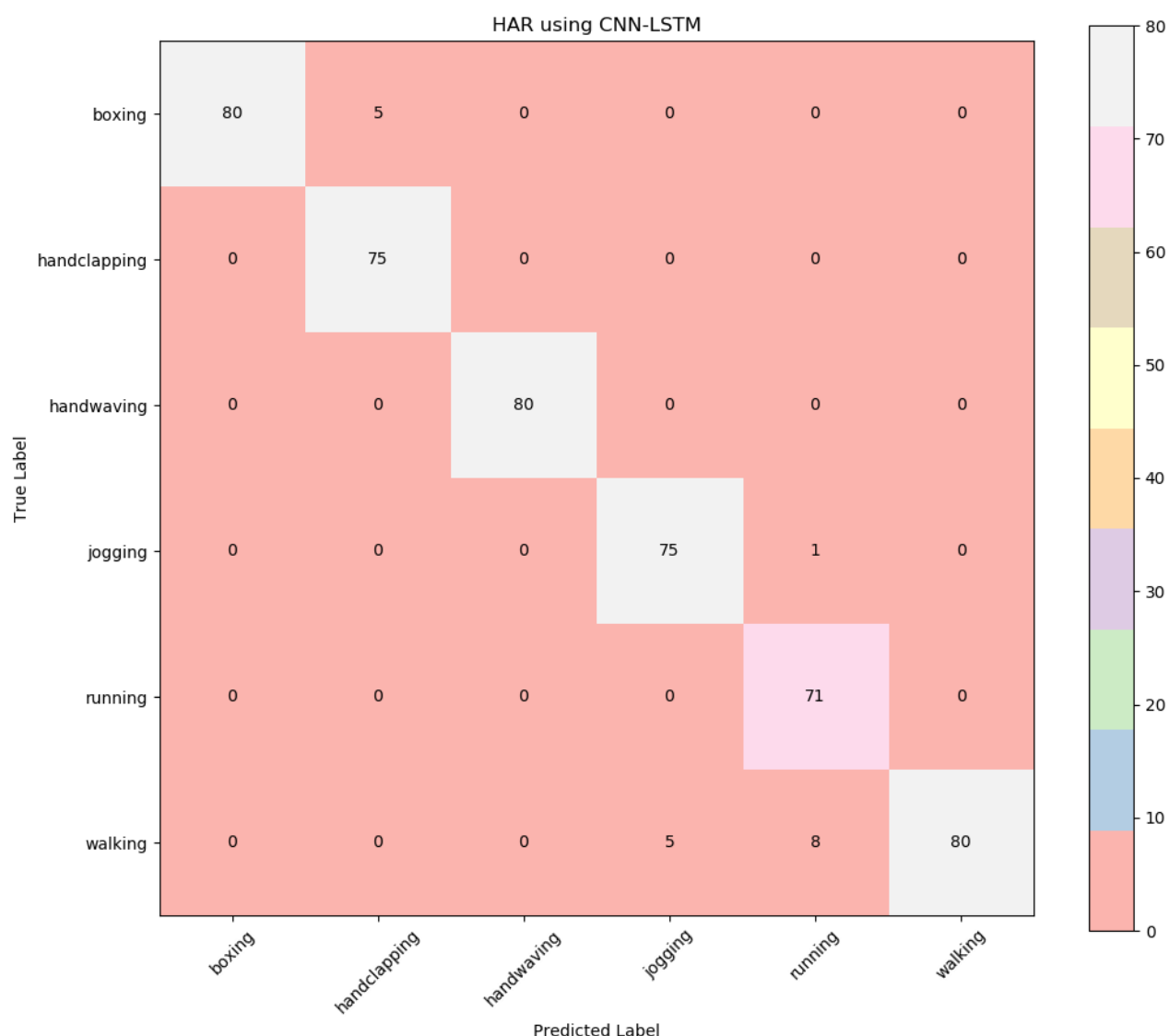


Figure 8.4 Sample Confusion Matrix Output for CNN-LSTM Combination

Figure 8.4 shows a sample confusion matrix of the CNN-LSTM combination. The predicted labels are expressed along the X-axis and the true labels are expressed along the Y-axis. According to the confusion matrix above, we observe that all 80 samples of boxing, handwaving and walking activities have been classified accurately, whereas 5 samples of handclapping have been wrongly classified as boxing, 5 samples of jogging have been wrongly classified as walking, and 9 samples of running have been wrongly classified as 1 for jogging and 8 for walking respectively.

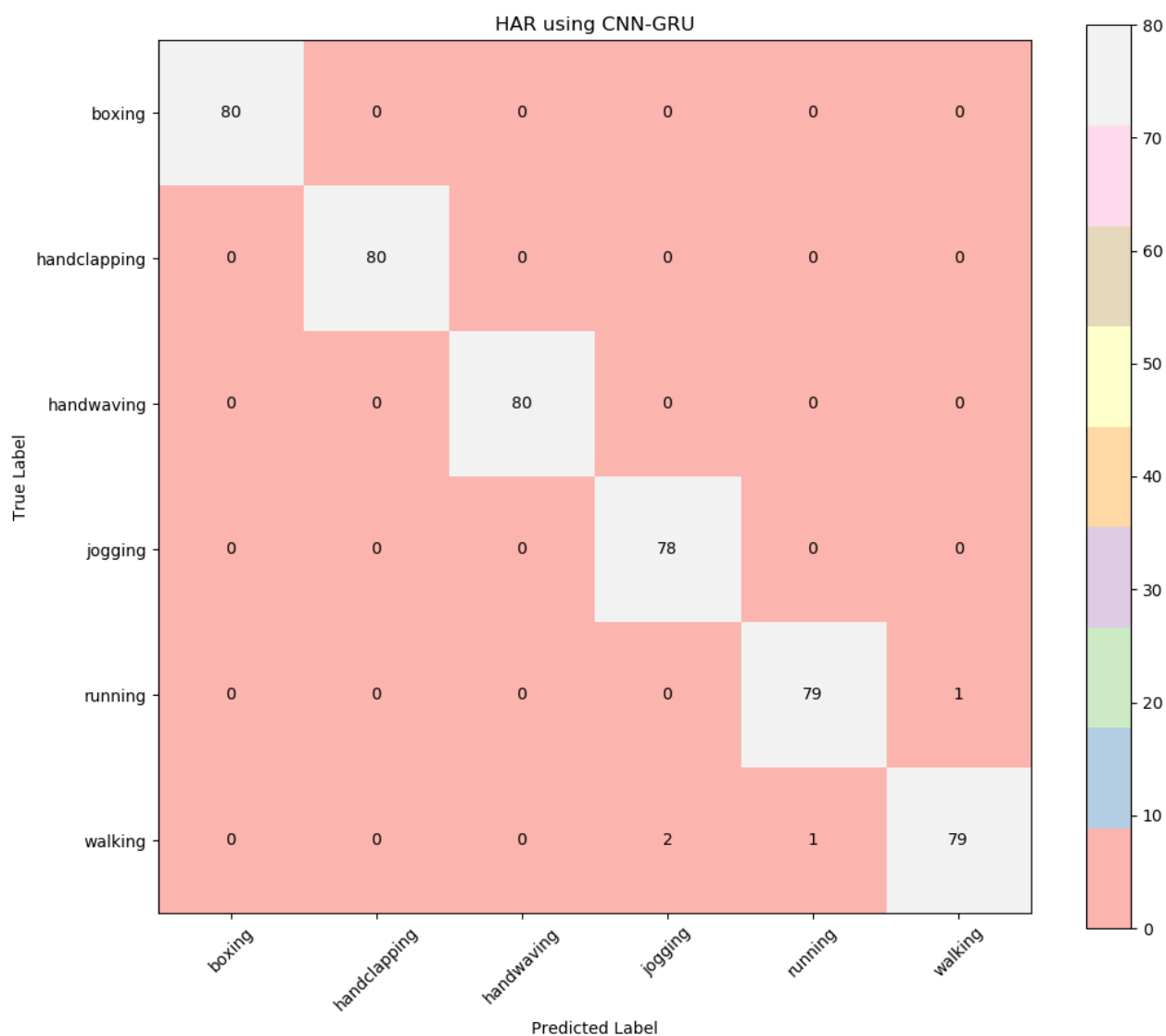


Figure 8.5 Sample Confusion Matrix Output for CNN-GRU Combination

Figure 8.5 shows a sample confusion matrix of the CNN-GRU combination. The predicted labels are expressed along the X-axis and the true labels are expressed along the Y-axis. According to the confusion matrix above, all 80 samples of boxing, handclapping and handwaving have been classified correctly, whereas 2 samples of jogging have been wrongly classified as walking, 1 sample of running has been wrongly classified as walking and 1 sample of walking has been wrongly classified as running.

# Conclusions and Future Work

Human Activity Recognition is one of the most sought-after concepts in recent times. The ability to recognize activities being performed based on a pre-defined dataset after employing machine learning and deep learning techniques is a great leap towards creating better machines for the future. There have been numerous attempts at doing these kinds of projects ever since the 1980s and the success of benchmarked and prototype systems always encourages researchers and students alike, to further study the concepts behind activity recognition, machine learning and deep learning.

In this project work, an attempt is made to recognize six simple activities – boxing, handclapping, handwaving, jogging, running and walking. While these may be normal day-to-day human activities, the study and understanding of concepts to propose a model to recognize these activities is in itself a challenging task. A combination of Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRUs) have made this work much easier. A sequential model was built based on multiple layers of these technologies and techniques, with the application of batch normalization to ensure that the network learns and dropout to avoid over-fitting of data.

The first part of performing a successful activity recognition task involves the accumulation and pre-processing of data. The KTH Dataset, being open-source and a collection of 600 videos, with 25 different persons performing the 6 activities in 4 different scenarios, made for a wide variety of data that could be used to conduct Human Activity Recognition. The conversion of the videos to frames was done with ease using the two different technologies – FFmpeg and OpenCV. In this project, it was observed that FFmpeg performed better than OpenCV in the conversion of videos to frames. Not only did it take significantly lesser time to convert the videos, it also consumed lesser memory in the secondary storage, that is, the local hard disk drive.

Building the sequential model involves a layered architecture that starts from the input layer, followed by multiple convolutional layers, dense layers, a flattening layer, activation layers, dropout layers and so forth. One of the main observations made here was the combination of these convolutional layers with two newer and upcoming recurrent unit technologies, the LSTM and GRU recurrent units. These provide for a sequence-to-sequence architecture and can be used in a wide variety of machine learning and deep learning tasks. In this project, an attempt has been made to conduct a comparative analysis of these two units and provide a brief conclusion as to which one



performed better. It was observed that for smaller training datasets, GRU performed better than LSTM due to the fact that it contains lesser parameters and a lack of an output gate means that more parameters can be used to perform classification. On the other hand, LSTM achieves significantly higher accuracy when the entire dataset is taken into consideration, which is more important. Not much has been known about GRU so far and researches are being made as to improve it for bigger and vast datasets.

Future work can be implemented to a wide extent in this aspect of human activity recognition. The inclusion of the KTH Dataset is meant for a simple and clear understanding of the concepts and techniques used to easily perform Human Activity Recognition. Much more complex architectures can be built to perform Human Activity Recognition to further enhance the accuracies obtained. Datasets can be obtained which contain more activities, have 3-channel RGB data and have better resolution when compared to KTH, and these can be used to perform activity recognition.

Furthermore, the enhancement of this particular combination of two or more techniques, like Recurrent Neural Networks (RNNs), Support Vector Machines (SVMs), shape moments, normalized Fourier descriptors, kNN classifiers, Spatio-Temporal Feature Learning (STFL) and sparse motion trajectories, can be used to perform Human Activity Recognition on a larger scale.

From an industry perspective, the application of a much more complex version of human activity recognition can be made by embedding the software into closed-circuit TVs (CCTVs) for the detection and recognition of any miscreants indulging in criminal activities. There is a possibility of the reduction in the crime rate if such a system is implemented in the future, which can detect crime even before it is committed, by performing activity recognition and analysis of day-to-day human activities.

## References

- [1] Shikhar Shrestha, “*Machine Learning for Human Activity Recognition from Video*”, Stanford University, CA, 2016.
- [2] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar and Li Fei-Fei, “*Large-scale Video Classification with Convolutional Neural Networks*”, Google Research, Stanford University, CA, 2014.
- [3] Christian Schödl, Ivan Laptev and Barbara Caputo, “*Recognizing Human Actions: A Local SVM Approach*”, Computational Vision and Active Perception Laboratory (CVAP) Dept. of Numerical Analysis and Computer Science KTH, SE-100 44 Stockholm, Sweden, 2004.
- [4] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani and Manohar Paluri, “*Learning Spatio-temporal Features with 3D Convolutional Networks*”, IEEE, 2015.
- [5] Nitish Srivastava, Elman Mansimov and Ruslan Salakhutdinov, “*Unsupervised Learning of Video Representations using LSTMs*”, University of Toronto, 2016.
- [6] Yang Qin, Lingfei Mo, Jing Ye and Zhening Du, “*Multi-channel Features Fitted 3D CNNs and LSTMs for Human Activity Recognition*”, Tenth International Conference on Sensing Technology, 2016.
- [7] Andros Tjandra, Sakriani Sakti, Ruli Manurung, Mirna Adriani and Satoshi Nakamura, “*Gated Recurrent Neural Tensor Network*”, IEEE, 2016.
- [8] Xuanhan Wang, Lianli Gao, Jingkuan Song and Hengtao Shen, “*Beyond Frame-level CNN: Saliency-aware 3D CNN with LSTM for Video Action Recognition*”, IEEE, 2016.
- [9] Nicolas Ballas, Li Yao, Chris Pal and Aaron Courville, “*Delving Deeper Into Convolutional Networks For Learning Video Representations*”, ICLR, 2016.
- [10] Fadwa Al-Azzo, Chunbo Bao, Arwa Mohammed Taqi, Mariofanna Milanova and Nabeel Ghassan, “*Human Actions Recognition Based on 3D Deep Neural Network*”, Annual Conference on New Trends in Information & Communications Technology Applications - NTICT 2017.
- [11] Wen-Hui Chen, Carlos Andrés Betancourt Baca and Chih-Hao Tou, “*LSTM-RNNs Combined with Scene Information for Human Activity Recognition*”, IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom), 2017.

- [12] Rahul Dey and Fathi M. Salem, “*Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks*”, IEEE, 2017.
- [13] John See and Saimunur Rahman, “*On the Effects of Low Video Quality in Human Action Recognition*”, IEEE, 2015.