

# **Healthcare monitoring system based on Deep Learning**

Enroll. No. (s)	- 18103316, 18103132, 18103139
Name of Student (s)	- Rahul Singhal, Himanshu Joshi, Pranjal Chandel
Name of supervisor	- Prof. Satish Chandra



**Department of Computer Science and Engineering**

**Jaypee Institute of Information Technology**

**May 2022**

## **TABLE OF CONTENTS**

<b>Chapter No.</b>	<b>Topics</b>	<b>Page No.</b>
<b>Chapter-1</b>	<b>Introduction</b>	9 to 12
	1.1 General background	
	1.2 Main objective	
	1.3 Importance of the problem	
	1.4 Empirical Research	
<b>Chapter-2</b>	<b>Literature Survey</b>	13 to 16
	2.1 Integrated summary of the literature studied	
<b>Chapter 3:</b>	<b>Requirement Analysis and Solution Approach</b>	17 to 21
	3.1 Requirement Analysis	
	3.1.1 Software requirements	
	3.1.2 Python libraries used in the project	
	3.1.3 Dataset used	
	3.1.4 Dataset Preparation	
	3.2 Brief description of the solution approach	
<b>Chapter-4</b>	<b>Modeling Implementation and Risk analysis</b>	22 to 40
	4.1 Proposed Methodology Workflow	
	4.1.1 Data Preprocessing	
	4.1.2 Data Visualization	
	4.1.3 Model Implementation	
	4.2 Risk Analysis and Mitigation	

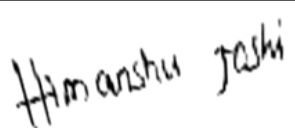
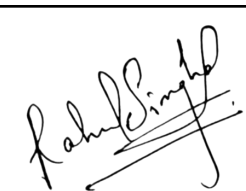
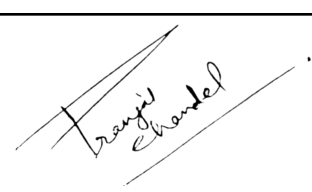
<b>Chapter-5</b>	<b>Testing</b>	41 to 44
	5.1 Testing plan and Components decomposition and type of testing required	
	5.2 Error and Exception Handling	
<b>Chapter-6</b>	<b>Results, Conclusion, and Future Work</b>	45 to 49
	6.1 Results	
	6.2 Conclusion, Limitations and Future Work	
<b>References</b>		50 to 51

## DECLARATION

We henceforth certify that this report is our own effort and that it includes no content previously released or authored by some other individual, nor content that has been acknowledged for the award of any other degree or qualification by any other higher education institution or university, except where due acknowledgement has been done in the report.

**Place: Noida**

**Date: May 2022**

Student ID	Student Name	Student signature
18103132	Himanshu joshi	
18103316	Rahul Singhal	
18103139	Pranjal Chandel	

**Signature (Supervisor)**

## CERTIFICATE

This is to certify that the work titled “Healthcare monitoring system based on Deep Learning” submitted by “Rahul Singhal, Himanshu Joshi, Pranjal Chandel” in partial fulfillment for the award of degree of B.Tech. of Jaypee Institute of Information Technology, Noida has been carried out under my supervision.

Signature of Supervisor

Name of Supervisor	Prof. Satish Chandra
Designation	Professor
Date	24.05.2022

# ACKNOWLEDGEMENT

First of all, we would like to express our gratitude towards the Almighty for enabling us to complete this mid-sem report on “Healthcare monitoring system based on Deep Learning”.

Also, most importantly, we would like to extend our sincere thanks towards our Project Supervisor Dr. Satish Chandra. His supervision and guidance shaped this report to be completed successfully. While working on this project, we are able to increase our knowledge and skills immensely.

The details of the group members are as follows:

NAME	ENROLLMENT NO.	BATCH	Personal Email ID	College Email Id
Rahul Singhal	18103316	B10	rahulsinghal1904@gmail.com	18103316@mail.jiit.ac.in
Himanshu Joshi	18103132	B4	himanshujit2018@gmail.com	18103132@mail.jiit.ac.in
Pranjal Chandel	18103139	B4	pchandel2000@gmail.com	18103139@mail.jiit.ac.in

## **LIST OF FIGURES**

- Figure 1: Architecture of the proposed methodology
- Figure 2: Testing : Snapshots of pictures
- Figure 3: Testing : Snapshots of pictures
- Figure 4: Three dimensional structure depicting 3D Convolution
- Figure 5: Max Pooling 3D Operation on a 3D convoluted output
- Figure 6: Network architecture of C3D
- Figure 7: Model structure of C3D Model
- Figure 8: Snapshot of Resnet18.txt
- Figure 9: Snapshot of Resnet50.txt
- Figure 10: Snapshot of Resnet101.txt
- Figure 11: Model structure of Resnext101
- Figure 12: Model Structure of Squeezenet
- Figure 13: Model Structure of MobileNetV2
- Figure 14: Model structure of ShuffleNetV2
- Figure 15: Some frames from the dataset showing ambiguity
- Figure 16: Comparative accuracy given by independent model and pretraining incorporated model
- Figure 17: Top one and Top two Accuracies
- Figure 18: Precision score of Resnet101 model for detecting different action categories
- Figure 19: Recall score of Resnet101 model for detecting different action categories
- Figure 20: F1-score of Resnet101 model for detecting different action categories

## **LIST OF TABLES**

Table 1: Integrated summary table of the literature studied

Table 2: Types of testing done

Table 3: Table showing Top one and Top two accuracies

Table 4: Evaluation metric Scores -Resnet101 for detecting different actions

Table 5: Evaluation metric Scores - Resnext101 for detecting different actions

Table 6: Precision, Recall and F1 Scores of Resnet50 for detecting different actions



# **CHAPTER 1**

## **1.1 General Background**

Nowadays, healthcare problems among elders have been increasing at an unprecedented rate and every year, more than a quarter of the elderly people face weakening injuries such as unexpected falls, etc. resulting in broken bones and serious injuries in some cases. Sometimes, these injuries may go unnoticed, and the resulting health consequences can have a considerable negative impact on quality of life. Constant surveillance by trained professionals is impossible owing to the expense, effort.

In this project, we plan to use nine medically relevant activities to train a neural network for action identification in patients. For a variety of reasons, classifying a specific action is difficult. Some of the activities differ just slightly from one another and, in many cases, are very similar. The only difference between the films in action classes for some classes is a tiny variation in hand positioning. Another issue is that the algorithms are limited to simply utilizing raw RGB photos to forecast activity.

## **1.2 Main objective**

To use nine medically relevant activities to train state of the art deep neural networks for action identification in patients. A video clip of a human action is fed into our neural network, and the result is the identification of that particular activity.

## **1.3 Importance of the problem**

This topic is gaining interest due to the rising needs in a variety of fields. HAR is used in healthcare systems to monitor activities of daily living, security environments for automatic detection of aberrant actions and notification of appropriate authorities, and to improve human-computer interaction.

Human activity recognition is critical for health and wellness applications as well as wearable healthcare systems. In order to recognise users' behaviors, these intelligent systems reason through vast volumes of sensor-derived data. Ground-truth data of adequate quality and quantity is needed for rigorous training and validation of action recognition algorithms. Ground truth is frequently obtained through video recordings, which can offer precise data when properly labeled. However, video annotation is a difficult undertaking that is subjective by definition. Furthermore, the researchers must keep the sensitive nature of the recordings in mind in order to preserve participants' identities and privacy.

Healthcare needs have altered dramatically in recent years. An ageing population, as well as an increase in chronic illnesses such as diabetes, obesity, cardiovascular disease, and neurological disorders, has prompted research into remedies. Thanks to technological improvements and rapid advances in artificial intelligence and the Internet of Things, low-cost sensors and sensing devices have become widely available. These innovations build on previous research in the areas of Ambient Intelligent Systems and Activity Recognition, as well as its applications in healthcare.

The selection of proper hardware, infrastructure design, software design, and other challenges face the design and implementation of these applications. It also covers machine learning algorithm training and validation, as well as system testing and deployment and proper information sharing. Each of these processes necessitates taking into account the demands and preferences of certain stakeholders. These ambient intelligent systems handle enormous amounts of sensor-derived data and require reliable and effective data mining algorithms to recognise behavior of interest in order to monitor health, well-being, or other personal advantages like fitness level. The output of these algorithms is generally compared with ground-truth or benchmark data to validate these tactics. Studies on specific characteristics of activity, such as motions, transitional activities, bodily postures, or specific disorders, are frequently conducted. As a result, labels for the second stage are frequently research-specific and not universally applicable. The data must then be annotated or transformed.

## **1.4 Empirical Research**

### **Spatial transform**

Max Jaderberg et al. presented Spatial Transformer components which are a prominent strategy for improving a model's spatial regularization across spatial transformations including translation, scaling, rotation, resizing, and non-rigid deflections. They may be added into existing convolutional networks in two ways: directly after the input or in further layers. They accomplish spatial invariance by adapting their input to a canonical, anticipated stance, which improves classification performance. The word adaptive means that a suitable transformation is generated for each sample, based on the input. Backpropagation may be used to train spatial transformer networks from start to finish.

### **Image Preprocessing and Augmentation**

A sequence of deterministic processes that normalize or enhance desirable data attributes make up image preprocessing. You can, for example, normalize data to a specified range or shrink data

to fit the network input layer's requirements. For training, validation, and test data, preprocessing is used.

Data augmentation is the process of applying randomized operations to the training data while the network is learning.

Variations in picture capture can be simulated using augmented image data. Random transformations such as rotation and translation, which initiate fluctuations in the camera orientation with regard to the scene, are common forms of picture augmentation procedures. Cropping at random simulates changes in scene composition. Artificial noise is used to replicate aberrations that occur during picture capture or data processing. Augmentation serves to make the network invariant to frequent fluctuations and distortions in the data by increasing the effective quantity of training data.

## Temporal Random Cropping

This is accomplished by randomly cropping the specified frame indices in time. If the size is less than the number of frames, loop the indices as many times as required to meet the size. Size is used as the parameter for temporal random cropping since it determines the crop's intended output size.

## Random Horizontally Flipping

A random horizontal flip is a type of visual data enhancement wherein a given image is horizontally reversed with a predetermined probability. Images are randomly rotated with a frequency of 0.5 in this project.

## Multi-Scale Corner Cropping

Crop the provided image to a size that is chosen at random. Using scales of the original length, a crop of size is chosen. Cropping positions are chosen at random from four corners and one center. This crop has been scaled to the specified size. Scales (actual size cropping scales), size (size of the shorter edge), and interpolation are all important attributes.

## Composing

This aids in the composition of many transformations. The transformations parameter is utilised, and it returns a list of transforms to combine.

## Scaling

Helps in rescaling the input image to the given size. Parameter used is size (sequence or int): Desired output size. The output dimension will be suited to size if it is like (w, h). If size is an integer, the picture's shorter side will be resized to (size \* height / width, size), i.e. if height > width, the image would be resized to (size \* height / width, size).

## Center Cropping

Cropping an image to the center is made easier. Size (sequence or int) is one of the parameters: Crop output size that is desired. A square crop (size, size) is created when size is an int rather than a sequence like (h, w).

## Jester dataset

The Jester gesture recognition dataset contains 148,092 video clips of individuals doing basic, pre-defined hand movements before a webcam or webcam that have been tagged. It's made to teach machine learning models how to detect human hand motions like swiping left or right, dragging two fingers down, and drumming fingers. The movies are divided into 27 separate classes of human hand signals and are used for training, development, and testing in an 8:1:1 ratio. Two "no gesture" classes are included in the dataset to aid the network in distinguishing between specified motions and unidentified hand motions. The relevance of gesture/action detection in human-computer interactions has expanded inside the age of mobile computing. The Jester video dataset enables machine learning algorithms to be trained to identify human hand gestures. More than 1,300 different crowd actors were used to build the Jester dataset.

## CHAPTER 2

### Literature Review

#### 2.1 Integrated summary of the literature studied

S. No.	Research Paper	Authors	Description	Technique used	Dataset	Future Work
1.	Human interaction recognition in youtube video	S CHO S.LIM ,PAR K,KW AK	Recognition of Human Activity in youtube video	Annotation tag	Over 200 Youtube video	Expansions of dataset and Generalize relevant tag
2	Who is doing what?Action and actors recognition	Mohammad sahnaz cheena	Recognition of actions of actors. The proposed method gave an accuracy of 93.1%	Bilinear model	IXMAS	Extend to action ,view ,actors
3	Recognition of complex human interaction	B zang, Yan yam	Recognizing how camera view change in television	Motion interchange pattern	TVHI	Extend to large dataset
4	HAR tracking using video	Robert bodor, Benet	Tracking of pedestrian outdoor using kalman filter	Optimal flow ,kalman filter,HMM	Real time traffic	Mobile tracking system for human activity
5	Study of verbalizing behavior in video	Chiro, mami naumi	Verbal words used for retrieving scene	Bag of word	Simple activity of offices	Some network to detect human activity

6	Real time crowd motion analysis	Nasim and chabane	Detection of abnormal activity in crowd	POI and Heat map	Escalator exit In airport	Acceleration by tracking POI
7	Simultaneous tracking and action recognition	W.Lu and j Little	Recognition of actions of athlete	Hog and pca	athletes	Actions as a random variable
8	Human activity prediction ongoing activity	Bo jhang yan yan	Early prediction of unfinished activity from the video	Integral bag of words and dynamic bag of words	UT interaction	Early recognition from the video
9	Human Activity Recognition for Video Surveillance	Weiya o Lin, Ming-Ting Sun, Radha Poovandran, and Zhengyou Zhang	Presents an innovative method for recognising people's actions using video data automatically.	Category Feature Vectors (CFVs), GMM, Confident-Frame-based Recognizing algorithm (CFR)	PETS'04 database	Incorporated deep learning and fine tune model to improve results
10	An Effective Approach for Human Activity Classification Using Feature Fusion and Machine Learning Methods	Muhammad Junaid Ibrahim, Jaweria Kainat, Hussain AlSalman, Syed Sajid Ullah	Using feature integration and machine learning approaches, the paper provides an efficient automated strategy.	SVM, KNN, Complex Tree	Weizman n , KTH Dataset	Use Deep learning to learn from a dataset better.

		,Suheer Al-Hadhrami and Saddam Hussain				
11	Learning spatiotemporal features with 3d convolutional networks	Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, Manohar Paluri	Proposed a straightforward yet successful method for acquiring temporal information based on deep three dimensional CNN trained on a huge supervised datasets collected.	C3D	UCF101 dataset	Increase accuracy by fine tuning the model parameters.
12	Resource Efficient 3D Convolutional Neural Networks	Okan Koyuklu, Neslihan Kose, Ahmet Gunduz, Gerhard Rigoll	Resnext 101 gave an accuracy of 68.3% on Kinetics-600 dataset, 94.89 on Jester dataset and 89.08 on UCF101 dataset.	Efficient 3D CNNs	Kinetics-600 dataset, Jester dataset, UCF-101 dataset	Increase accuracy by hyperparameter tuning
13	Activity Recognition with Still Images and	Wendell Hom	On a fifteen classes subset, a	2D CNN, C3D, ResNeXt-10	Kinetic 600 dataset	3D NN gave 35.2% while Resnext 101

	Video		27-category subset, and ultimately the whole six hundred Kinetics dataset, a 2D CNN, 3D CNN, and a deep 3D CNN was implemented.	1		gave 24.4% accuracy on Kinetic dataset
--	-------	--	---	---	--	--

Table 1. Literature review



## **CHAPTER 3**

### **3.1 Requirement Analysis**

#### **3.1.1 Software requirements**

The code has been implemented in Python 3 in Google colab and PyCharm in Windows environment.

- 1) Python 3
- 2) Google Colab
- 3) Pycharm
- 4) Visual studio code

#### **3.1.2 Python Libraries used for the project**

The libraries used in the code are:

Os

This package gives us the ability to run and implement the functions related to the operating system. The os module also provides extensions specific to a given operating system; however, employing them compromises portability. Bytes and string objects are accepted by all functions that take path or file names.

Numpy

Numpy is a Python package that includes an array of objects of more than one dimension, derivative objects, and has various methods. The NumPy representation, indexing, and transmission principles are today's de-facto array computing standards because they are fast and adaptable.

## Pandas

Pandas is quick and provides users with excellent performance and productivity. Wes McKinney, who worked at AQR Capital Management at the time, invented Pandas in 2008. Many variants of pandas have indeed been developed over time. 1.4.1 is the most recent version of the pandas.

## Sklearn

Sklearn was first developed as a GSOC project in 2007 by David Cournapeau under the name scikits.learn and is used to implement ML related operations.

## Copy

This module handles shallow and deep copy operations in general.

## Math

This module provides various functionalities. Except as otherwise specified, all return values remain floats.

## Seaborn

Seaborn has features such as a dataset-oriented API for determining the connection between variables, as well as other features. It includes high-level abstractions for multi-plot grids, displaying univariate and multivariate distributions, and automatic estimating and drawing of linear regression plots.

## Shutil

It copies the information of the fsrc file-like object to the fdst file-like object. The buffer size is determined by the integer length, if one is provided. A negative length number, in particular, signifies that the data is copied without looping over the source data in chunks; the data is read in chunks by default to minimize uncontrolled memory usage. Only the information from the existing file location to the end of the document will be duplicated if the existing file location of the fsrc object is not 0.

## TorchvisionCsv

(i) Stable: These functionalities will be supported in the long run, and there should be no significant performance issues or documentation gaps. Backwards compatibility is also expected.

(ii) Beta: Because the API may change depending on user input, performance has to be improved, or coverage across operators is not yet complete, features are labeled as Beta. When it comes to Beta features, we're committed to seeing them through to Stable status. However, we will not guarantee backwards compatibility.

(iii) Prototype: These capabilities are normally not included in binary distributions such as PyPI or Conda, unless hidden behind run-time options, and are still in the early stages of development for feedback and testing.

## Sys

This package gives access to several variables utilized or handled by the interpreter. It is readily accessible. This includes the ABI flags provided by PEP 3149 for POSIX platforms when Python was generated with the normal configure script.

## Pathlib

Path is most likely what you need if you've never used this module before or aren't sure which class is best for your work. It creates a concrete route for the platform on which the code runs.

Pure pathways are beneficial in a few specific situations, such as:

On a Unix system, you can alter Windows paths (or vice versa). When operating on Unix, you can't create a WindowsPath, but you can create a PureWindowsPath.

## Functools

Higher-order functions, or functions that operate on or yield other functions, are handled by the functools module. For the functionality of this module, any callable entity can be considered as a function. It contains functions such as `@functools.cache(user function)` Unbounded function cache, simple and lightweight. Also known as "memoize." Returns the same as `lru cache(maxsize=None)`, wrapping a dictionary search for the function parameters in a thin wrapper. This is smaller and quicker than `lru cache()` with a limit set since it never has to evict old data

## Joblib

Joblib is a suite of Python utilities for lightweight pipelining. This involves transparent disc caching of functions and delayed re-evaluation are two examples (memoize pattern) parallel computing made simple. The license is BSD. Joblib solves problems while preserving as much of your code and flow control as feasible.

A memoize or make-like tool for Python functions which works well for random Python objects, even very big numpy arrays. Write the operations as a collection of stages with well-defined inputs and outputs to differentiate persistence and flow-execution reasoning from domain reasoning or algorithmic code. Joblib's computations can be saved to disc and repeated only if required.

## Vision

The goal is to give tools that make it simple to improve performance and consistency when working on long-running projects.

Avoid repeating computations: code is frequently rerun, for example, when prototyping computationally intensive activities (as in scientific research), but hand-crafted methods to address this problem are error-prone and frequently result in unreproducible outcomes.

Straightforwardly persist to disc: effectively preserving random entities carrying huge data is difficult.

### 3.1.3 Dataset used

This dataset was taken from NTU Singapore and it contains 60 action classes and 56,880 video samples. We concentrate on a subset of 9 healthcare related action types. and fanning self are the action classes we focus on. We extracted videos according to our need from the NTU RGB+D dataset.

### 3.1.4 Dataset Preparation

- First, we convert from avi to jpg files
- Generate n\_frames files from the given giles
- Generate annotation file in json format where annotation directory path includes will include classInd, training list, testing list

## 3.2 Steps taken to implement our project

- First of all, we Installed the necessary libraries which have been mentioned before in the report.
- Then data was collected for specific action classes from NTU RGB dataset for activity recognition.
- We preprocessed the data collected and performed data manipulation to extract relevant information and remove redundant data to enhance performance.
- Then we generated the jpg files from avi videos and processed them to produce the json files and n frames files.
- Data visualization was done to understand the data better and gather useful insights.
- Then we trained all the models on the data to learn from the training data.
- Validation of all models was done using the NTU RGB dataset.
- Testing was performed on the collected data.
- Performance of models was evaluated using metrics such as accuracy, precision, recall, confusion matrix.
- Tuning of hyperparameters was done for learning rate, number of epochs, etc.

## CHAPTER 4

### Modeling Implementation and Risk Analysis

#### 4.1 Proposed Methodology Architecture

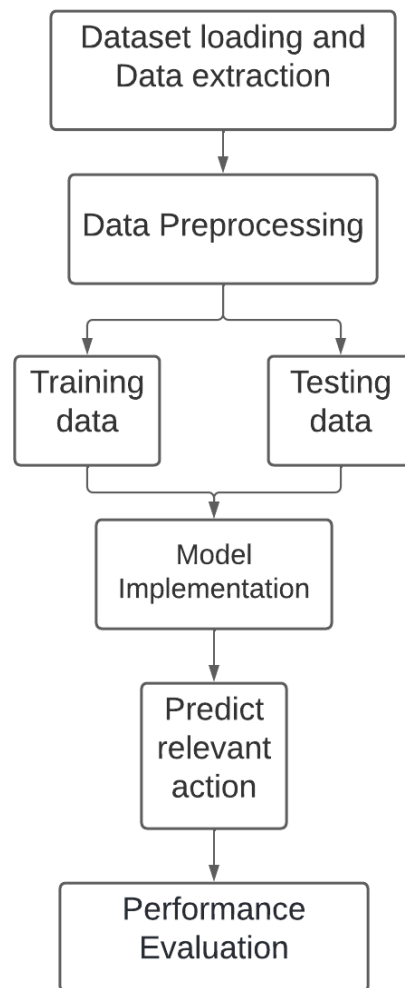


Figure 1. Architecture of the proposed methodology

### 4.1.1 Data pre-processing

All films are broken into individual jpg frames and scaled down to four hundred twenty pixels width and two hundred forty pixels pixels as part of the pre-processing. Every clip has three channels and has M frames, where M is the count of frames per movie, which might range from thirty three to 222. The dataset is pre-processed in both geographical and temporal dimensions. To eliminate superfluous background information, every video is made to two hundred forty pixels each side after resizing them.

Cropping does not result in the elimination of the target topic, hence the scales are determined by us evaluating fifty video samples. We use bilinear interpolation to spatially enlarge the generated pictures to 112 by 112 pixels after cropping.

Important functions used for data preprocessing were:

- Filtering the directory, Index Mapping and Extracting the labels

```
x
['headache/S001C001P001R001A044_rgb',
 'headache/S001C001P002R001A044_rgb',
 'fallingDown/S001C001P001R001A043_rgb',
 'fallingDown/S001C001P002R001A043_rgb',
 'sneezeCough/S001C001P001R001A041_rgb',
 'sneezeCough/S001C001P002R001A041_rgb',
 'backPain/S001C001P002R001A046_rgb',
 'backPain/S001C001P001R001A046_rgb',
 'fanSelf/S001C001P001R001A049_rgb',
 'fanSelf/S001C001P002R001A049_rgb',
 'staggering/S001C001P002R001A042_rgb',
 'staggering/S001C001P001R001A042_rgb',
 'nauseaVomiting/S001C001P001R001A048_rgb',
 'nauseaVomiting/S001C001P002R001A048_rgb',
 'chestPain/S001C001P001R001A045_rgb',
 'chestPain/S001C001P002R001A045_rgb',
 'neckPain/S001C001P002R001A047_rgb',
 'neckPain/S001C001P001R001A047_rgb']

y
[44, 44, 43, 43, 41, 41, 46, 46, 49, 49, 42, 42, 48, 48, 45, 45, 47, 47]
```

- Create the classInd, train list and test list

classInd.txt ×

```
1 41 sneezeCough
2 42 staggering
3 43 fallingDown
4 44 headache
5 45 chestPain
6 46 backPain
7 47 neckPain
8 48 nauseaVomiting
9 49 fanSelf
```

trainlist01.txt ×

```
1 snz/S015C003P008R002A041 41
2 snz/S015C003P015R001A041 41
3 snz/S015C002P025R002A041 41
4 snz/S015C002P019R001A041 41
5 snz/S015C003P025R002A041 41
6 snz/S015C001P019R002A041 41
7 snz/S015C001P019R001A041 41
8 snz/S015C003P019R001A041 41
9 snz/S015C002P016R001A041 41
10 snz/S015C002P016R002A041 41
11 snz/S015C003P007R001A041 41
12 snz/S015C001P016R001A041 41
13 snz/S015C002P008R001A041 41
14 snz/S015C003P016R001A041 41
15 snz/S015C001P025R002A041 41
16 snz/S015C002P008R002A041 41
17 snz/S015C002P025R001A041 41
18 snz/S015C003P017R001A041 41
19 snz/S015C001P008R002A041 41
20 snz/S015C001P037R001A041 41
21 snz/S015C003P007R002A041 41
22 snz/S015C003P008R001A041 41
23 snz/S015C003P016R002A041 41
24 snz/S015C002P017R001A041 41
```

testlist01.txt ×

```
1 "Sneeze or Cough/S015C003P037R002A041" 41
2 "Sneeze or Cough/S015C002P019R002A041" 41
3 "Sneeze or Cough/S015C001P008R002A041" 41
4 "Sneeze or Cough/S015C002P017R001A041" 41
5 "Sneeze or Cough/S015C003P008R002A041" 41
6 "Sneeze or Cough/S015C003P025R001A041" 41
7 "Sneeze or Cough/S015C003P025R002A041" 41
8 "Sneeze or Cough/S015C001P008R001A041" 41
9 "Sneeze or Cough/S015C001P025R002A041" 41
10 Staggering/S015C003P017R002A042 42
11 Staggering/S015C003P015R001A042 42
12 Staggering/S015C001P037R001A042 42
13 Staggering/S015C003P025R001A042 42
14 Staggering/S015C001P019R002A042 42
15 Staggering/S015C001P008R001A042 42
16 Staggering/S015C001P007R001A042 42
17 Staggering/S015C001P007R002A042 42
18 Staggering/S015C001P017R002A042 42
19 Staggering/S015C001P017R001A042 42
20 "Falling down/S015C003P025R001A043" 43
21 "Falling down/S015C003P019R001A043" 43
22 "Falling down/S015C002P015R001A043" 43
23 "Falling down/S015C001P007R002A043" 43
```



- Converting jpg to json format

```
ntu_01.json X
1 {"labels": ["snz", "stag", "fall", "head", "check", "back", "neck", "naus", "fan"], "database": {"S015C003P019R001A041_rgb": {
  "S015C003P017R002A043_rgb": {"subset": "training", "annotations": {"label": "fall", "segment": [1, 50]}}, "S015C002P008R002A04
  "S015C001P019R002A045_rgb": {"subset": "training", "annotations": {"label": "chest", "segment": [1, 76]}}, "S015C001P025R002A0
  }, "S015C001P019R001A048_rgb": {"subset": "training", "annotations": {"label": "naus", "segment": [1, 115]}}, "S015C001P016R00
  [1, 118]}}, "S015C003P017R001A044_rgb": {"subset": "validation", "annotations": {"label": "head", "segment": [1, 82]}}, "S015C
```

- Getting class labels

```
print(class_names)

{0: 'sneezeCough', 1: 'staggering', 2: 'fallingDown', 3: 'headache', 4: 'chestPain', 5: 'backPain', 6: 'neckPain', 7: 'nauseaVomiting', 8: 'fanSelf'}
```


- Loading ground truth and printing its dimensions

```
im.show()

width, height = im.size

print(width, height)

427 240
```



- Scales

```
scales = [1.0]
for i in range(1, 5):
    scales.append(scales[-1] * 0.97)

scales

[1.0, 0.97, 0.9409, 0.912673, 0.8852928099999999]
```

- Processing on Training data (Multiscale Random Crop, Multi scale Corner Crop and Temporal Centre Crop)

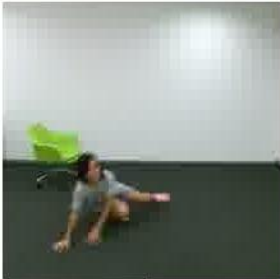
```
train_data('center', 16, 2, 112, scales)
```



[1.0, 0.97, 0.9409, 0.912673, 0.8852928099999999]  
scale0.912673  
427 240  
219



427 240  
219



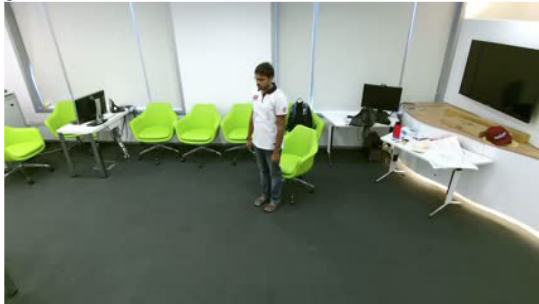
427 240  
219



- Processing on Testing data (Temporal Center Crop)

```
test_data(16, 2, 112)
```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,





### 4.1.2 Data Visualization

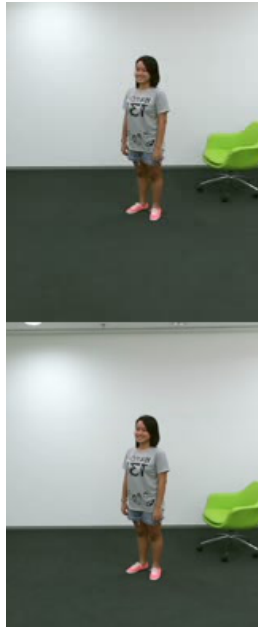


Figure 2. Training: Snapshots of pictures





Figure 3. Testing : Snapshots of pictures

### 4.1.3 Model Implementation

#### C3D

A three-dimensional volume space is the output shape. By wrapping the center of a cube and adding neighboring layers above, we may create 3D convolution.

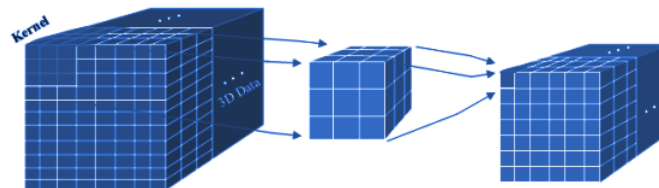


Figure 4. Three dimensional structure depicting 3D Convolution

The MaxPooling-3D layer is a three-dimensional nonlinear down sampling of a given object. It divides the input tensor data into three-dimensional sub- tensors and picks the sub-tensor object with the highest numeric value. Finally, it replaces each subtensor with its largest element to transform the input tensor to the output tensor.

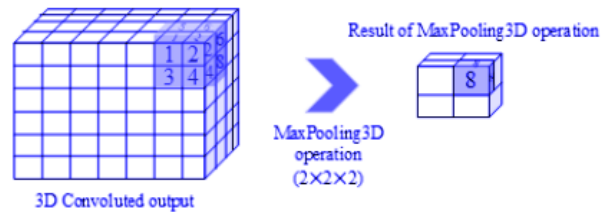


Figure 5. Max Pooling 3D Operation on a 3D convoluted output

The important layers are given below:

- MaxPooling3D: for compressing picture data. A stride is also specified, which determines how many pixels the filter travels in each iteration.
- Batch Normalization
- Dense layer
- Flatten layer

## C3D Architecture

As illustrated in figure below, the network architecture consists of eight convolutional layers, five pooling layers, and two fully connected layers.



Figure 6. Network architecture of C3D

The initial convolution layer is  $1 \times 3 \times 3$ , followed by a  $1 \times 2 \times 2$  pooling layer. This is done so that the temporal information is preserved in the first layer and higher-level representations of the temporal information may be built in the network's following levels. Every other convolution layer and pooling layer would be  $3 \times 3 \times 3$  and  $2 \times 2 \times 2$ , respectively, with strides of 1 and 2.

```

DataParallel(
  (module): C3D(
    (group1): Sequential(
      (0): Conv3d(3, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool3d(kernel_size=(2, 2, 2), stride=(1, 2, 2), padding=0, dilation=1, ceil_mode=False)
    )
    (group2): Sequential(
      (0): Conv3d(64, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=0, dilation=1, ceil_mode=False)
    )
    (group3): Sequential(
      (0): Conv3d(128, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (4): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=0, dilation=1, ceil_mode=False)
    )
    (group4): Sequential(
      (0): Conv3d(256, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (4): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=0, dilation=1, ceil_mode=False)
    )
    (group5): Sequential(
      (0): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (4): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool3d(kernel_size=(1, 2, 2), stride=(2, 2, 2), padding=(0, 1, 1), dilation=1, ceil_mode=False)
    )
    (fc1): Sequential(
      (0): Linear(in_features=8192, out_features=4096, bias=True)
      (1): ReLU()
      (2): Dropout(p=0.5, inplace=False)
    )
    (fc2): Sequential(
      (0): Linear(in_features=4096, out_features=4096, bias=True)
      (1): ReLU()
      (2): Dropout(p=0.5, inplace=False)
    )
    (fc): Sequential(
      (0): Linear(in_features=4096, out_features=9, bias=True)
    )
  )
)

```

Figure 7. Model structure of C3D Model

## Resnet Models 3D

ResNeXt is a homogeneous neural network that uses fewer hyperparameters than ResNet. This is done by the addition of a third dimension to ResNet's width and depth called "cardinality." The cardinality of a set of transformations determines its size.

ResNet 3D is a video model that makes use of 3D convolutions. There are two primary varieties in this model series.

1. The first benefit is that between these two procedures, there is an extra nonlinear rectification. When compared to a network utilizing complete 3D convolutions for the same number of



parameters, this essentially doubles the amount of nonlinearities, allowing the model to reflect more complicated functions.

2. The second possible benefit is that the deconstruction makes optimization easier, resulting in fewer training and testing losses in practice.

```
DataParallel(  
  (module): ResNet(  
    (conv1): Conv3d(3, 64, kernel_size=(7, 7, 7), stride=(1, 2, 2), padding=(3, 3, 3), bias=False)  
    (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (relu): ReLU(inplace=True)  
    (maxpool): MaxPool3d(kernel_size=(3, 3, 3), stride=2, padding=1, dilation=1, ceil_mode=False)  
    (layer1): Sequential(  
      (0): BasicBlock(  
        (conv1): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
      (1): BasicBlock(  
        (conv1): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
    )  
    (layer2): Sequential(  
      (0): BasicBlock(  
        (conv1): Conv3d(64, 128, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
      (1): BasicBlock(  
        (conv1): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
    )  
    (layer3): Sequential(  
      (0): BasicBlock(  
        (conv1): Conv3d(128, 256, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
      (1): BasicBlock(  
        (conv1): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
    )  
    (layer4): Sequential(  
      (0): BasicBlock(  
        (conv1): Conv3d(256, 512, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
      (1): BasicBlock(  
        (conv1): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)  
        (bn2): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
    )  
    (avgpool): AvgPool3d(kernel_size=(1, 4, 4), stride=1, padding=0)  
    (fc): Linear(in_features=512, out_features=9, bias=True)  
  )  
)
```

Figure 8. Snapshot of Resnet18.txt

```

DataParallel(
  (module): ResNet(
    (conv1): Conv3d(3, 64, kernel_size=(7, 7, 7), stride=(1, 2, 2), padding=(3, 3, 3), bias=False)
    (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool3d(kernel_size=(3, 3, 3), stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
        (conv1): Conv3d(64, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv3d(256, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv3d(256, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (layer2): Sequential(
      (0): Bottleneck(
        (conv1): Conv3d(256, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(2, 2, 2), bias=False)
          (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv3d(512, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv3d(512, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (3): Bottleneck(
        (conv1): Conv3d(512, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
  )
)

```

Figure 9. Snapshot of Resnet50.txt

```

DataParallel(
  (module): ResNet(
    (conv1): Conv3d(3, 64, kernel_size=(7, 7, 7), stride=(1, 2, 2), padding=(3, 3, 3), bias=False)
    (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool3d(kernel_size=(3, 3, 3), stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
        (conv1): Conv3d(64, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv3d(256, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv3d(256, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (layer2): Sequential(
      (0): Bottleneck(
        (conv1): Conv3d(256, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(2, 2, 2), bias=False)
          (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv3d(512, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv3d(512, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
  )
)

```

Figure 10. Snapshot of Resnet101.txt

```

DataParallel(
  (module): ResNeXt(
    (conv1): Conv3d(3, 64, kernel_size=(7, 7, 7), stride=(1, 2, 2), padding=(3, 3, 3), bias=False)
    (bn1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool3d(kernel_size=(3, 3, 3), stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): ResNeXtBottleneck(
        (conv1): Conv3d(64, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)
        (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(128, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
    )
    (1): ResNeXtBottleneck(
      (conv1): Conv3d(256, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
      (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)
      (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv3d(128, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
      (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): ResNeXtBottleneck(
      (conv1): Conv3d(256, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
      (bn1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)
      (bn2): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv3d(128, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
      (bn3): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (layer2): Sequential(
      (0): ResNeXtBottleneck(
        (conv1): Conv3d(256, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), groups=32, bias=False)
        (bn2): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(2, 2, 2), bias=False)
          (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): ResNeXtBottleneck(
        (conv1): Conv3d(512, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)
        (bn2): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): ResNeXtBottleneck(
        (conv1): Conv3d(512, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)
        (bn2): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (3): ResNeXtBottleneck(
        (conv1): Conv3d(512, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)
        (bn2): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv3d(256, 512, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
        (bn3): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
  )
)

```

Figure 11. Snapshot of Resnext101.txt

## Efficient Three dimensional Convolutional Neural Networks

They decrease computations by using group convolutions and depth wise separable convolutions. The MobileNetV2, ShuffleNetV2, and SqueezeNet designs are the focus of this project.



# Three dimensional-SqueezeNet

```
(module): SqueezeNet(
  (features): Sequential(
    (0): Conv3d(3, 64, kernel_size=(3, 3, 3), stride=(1, 2, 2), padding=(1, 1, 1))
    (1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool3d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(64, 16, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(16, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(16, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (5): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(128, 16, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(16, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(16, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (6): MaxPool3d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (7): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(128, 32, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(32, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(32, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (8): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(256, 32, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(32, 128, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(32, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (9): MaxPool3d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (10): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(256, 48, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(48, 192, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(48, 192, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (11): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(384, 48, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(48, 192, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(48, 192, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (12): MaxPool3d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (13): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(384, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(64, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (14): Fire(
      (relu): ReLU(inplace=True)
      (squeeze): Conv3d(512, 64, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (squeeze_bn): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand1x1): Conv3d(64, 256, kernel_size=(1, 1, 1), stride=(1, 1, 1))
      (expand1x1_bn): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (expand3x3): Conv3d(64, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (expand3x3_bn): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Conv3d(512, 9, kernel_size=(1, 1, 1), stride=(1, 1, 1))
    (2): ReLU(inplace=True)
    (3): AvgPool3d(kernel_size=(1, 4, 4), stride=1, padding=0)
  )
)
```

Figure 12. Model Structure of Squeezenet

## Three dimensional-MobileNetV2

```
DataParallel(  
  (module): MobileNetV2(  
    (features): Sequential(  
      (0): Sequential(  
        (0): Conv3d(3, 32, kernel_size=(3, 3, 3), stride=(1, 2, 2), padding=(1, 1, 1), bias=False)  
        (1): BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (2): ReLU6(inplace=True)  
      )  
      (1): InvertedResidual(  
        (conv): Sequential(  
          (0): Conv3d(32, 32, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=32, bias=False)  
          (1): BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (2): ReLU6(inplace=True)  
          (3): Conv3d(32, 16, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (4): BatchNorm3d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        )  
      )  
      (2): InvertedResidual(  
        (conv): Sequential(  
          (0): Conv3d(16, 96, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (1): BatchNorm3d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (2): ReLU6(inplace=True)  
          (3): Conv3d(96, 96, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), groups=96, bias=False)  
          (4): BatchNorm3d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (5): ReLU6(inplace=True)  
          (6): Conv3d(96, 24, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (7): BatchNorm3d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        )  
      )  
      (3): InvertedResidual(  
        (conv): Sequential(  
          (0): Conv3d(24, 144, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (1): BatchNorm3d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (2): ReLU6(inplace=True)  
          (3): Conv3d(144, 144, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=144, bias=False)  
          (4): BatchNorm3d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (5): ReLU6(inplace=True)  
          (6): Conv3d(144, 24, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (7): BatchNorm3d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        )  
      )  
      (4): InvertedResidual(  
        (conv): Sequential(  
          (0): Conv3d(24, 144, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (1): BatchNorm3d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (2): ReLU6(inplace=True)  
          (3): Conv3d(144, 144, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), groups=144, bias=False)  
          (4): BatchNorm3d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
          (5): ReLU6(inplace=True)  
          (6): Conv3d(144, 32, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)  
          (7): BatchNorm3d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        )  
      )  
    )  
  )  
)
```

Figure 13. Model Structure of MobileNetV2

## Three dimensional-ShuffleNetV2

```

DataParallel(
  (module): ShuffleNetV2(
    (conv1): Sequential(
      (0): Conv3d(3, 24, kernel_size=(3, 3, 3), stride=(1, 2, 2), padding=(1, 1, 1), bias=False)
      (1): BatchNorm3d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (maxpool): MaxPool3d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (features): Sequential(
      (0): InvertedResidual(
        (banch1): Sequential(
          (0): Conv3d(24, 24, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), groups=24, bias=False)
          (1): BatchNorm3d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Conv3d(24, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (3): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (4): ReLU(inplace=True)
        )
        (banch2): Sequential(
          (0): Conv3d(24, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(58, 58, kernel_size=(3, 3, 3), stride=(2, 2, 2), padding=(1, 1, 1), groups=58, bias=False)
          (4): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (6): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (7): ReLU(inplace=True)
        )
      )
      (1): InvertedResidual(
        (banch2): Sequential(
          (0): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(58, 58, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=58, bias=False)
          (4): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (6): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (7): ReLU(inplace=True)
        )
      )
      (2): InvertedResidual(
        (banch2): Sequential(
          (0): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(58, 58, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=58, bias=False)
          (4): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (6): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (7): ReLU(inplace=True)
        )
      )
      (3): InvertedResidual(
        (banch2): Sequential(
          (0): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (1): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(58, 58, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1), groups=58, bias=False)
          (4): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): Conv3d(58, 58, kernel_size=(1, 1, 1), stride=(1, 1, 1), bias=False)
          (6): BatchNorm3d(58, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (7): ReLU(inplace=True)
        )
      )
    )
  )
)

```

Figure 14. Model structure of ShuffleNetV2

## Training and Validation details

Categorical cross-entropy loss with stochastic gradient descent was employed for training the model. To optimize memory consumption and learning speed, we selected a batch containing sixty four videos. The epochs is fixed to fifty due to GPU constraints and the vast number of scheduled operations, although more epochs may increase performance. We played about with LR, settling on one thousandth for the initial forty iterations only until validation loss reaches the saturation point, then reducing it to 0.0001 for the final ten. We perform a center cropping to generate a squared picture, to one hundred and twelve pixels each side, and spatially resize the images to obtain the thirty two instants, and then downsample the picture for the validation set.

## 4.2 Risk Analysis and Mitigation

1. Because the method handles data files of enormous size, it takes a long time to run, and it's not unreasonable to label it sluggish. To operate efficiently, the model demands powerful GPUs.
2. The collecting of data files is an issue that arises since there are many different datasets accessible, each with its own set of outcomes, making it necessary to test for the optimal use case dataset. Data pre-processing and duplicate data removal are two key issues that arise while doing action recognition.
3. For a number of reasons, the action classification task is complex. To start with, some of the nine actions are quite comparable and have slight differences. The figure below shows how the action categories of headache, chest problem, and neck pain change merely in hand location.
4. The techniques are confined to forecasting behavior using just raw RGB pictures. This places further constraints on training and deployment.

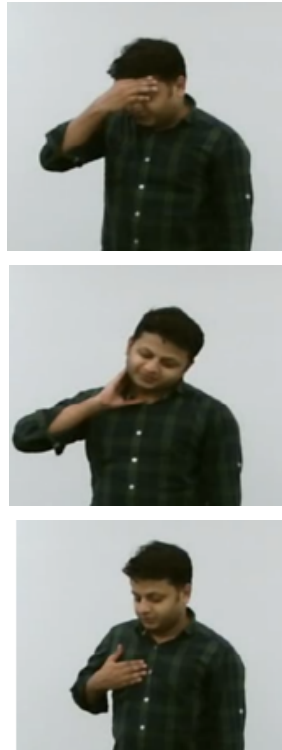


Figure 15. Some frames from the dataset showing ambiguity



## CHAPTER 5

### TESTING:

#### 5.1 Testing plan and Component decomposition and type of testing required:

The objectives behind the testing of our developed model are:

- Evaluation of Parameters of the developed system
- Calculating accuracy
- Efficiency of the model
- Evaluation of Complexity
- User Level Testing

Type of tests	Explanation	Software Component
Requirement Testing (Yes)	<p>The hardware and software specs were validated to confirm that they met the basic requirements.</p> <p>We also utilized the best GPU we could find (from Google Colab).</p>	VS Code/ Colab/ Jupyter Notebook /Python
Performance Testing(Yes)	<p>The process of assessing the suggested model's speed, accuracy, and consistency is known as performance testing.</p> <p>This was accomplished by developing, training, and testing the entire model using a variety of training approaches to get the fastest approach.</p>	VS Code/ Colab/ Jupyter Notebook /Python

Experimental Testing(Yes)	<p>To assure the best outcomes, we tested our model against a variety of experimental tests to fine-tune the hyperparameters.</p> <p>According to our resources, the best hardware specification was utilized.</p>	VS Code/ Colab/ Jupyter Notebook /Python
Unit Testing(Yes)	To guarantee legitimate and consistent findings, the output of the data preparation processes and the query result were both randomly checked in our project.	VS Code/ Colab/ Jupyter Notebook /Python
Integration Tests(Yes)	At regular intervals, tests were run to ensure that the new code did not interfere with previously written code.	VS Code/ Colab/ Jupyter Notebook /Python
Regression Tests(Yes)	This testing was performed in this project regularly to ensure that code in at different intervals does not cause any sort of problem to current programs or implementation.	VS Code/ Colab/ Jupyter Notebook /Python
Invariance Tests(Yes)	<p>It checks whether the various components of your machine learning workflow are compatible.</p> <p>Data augmentation is a commonly used strategy for testing invariance.</p>	VS Code/ Colab/ Jupyter Notebook /Python

	<p>You compare the effects of changed and unaltered input examples on the model output.</p> <p>We ran invariance testing on our dataset images before and after modifications to see if it results in any change in the output.</p>	
Minimum Functionality Test(Yes)	The basic functionality test allows you to determine whether actual model components perform as expected. The logic behind such tests is that total outcome based performance might hide crucial forthcoming model flaws.	VS Code/ Colab/ Jupyter Notebook /Python
Direction Expectational Tests(Yes)	To describe the predicted consequences of input distribution changes on the output, directional expectation tests were used. Seeing a different outcome might reflect incorrect assumptions about our input-output connection or dataset distribution.	VS Code/ Colab/ Jupyter Notebook /Python
Compliance Testing(Yes)	Conformance testing was used to establish how a system during a test validates that it meets the specific requirements of a standard. Compliance Testing is	VS Code/ Colab/ Jupyter Notebook /Python

	another name for conformance testing.	
--	--	--

Table 2: Types of testing done

## 5.2 Error and Exception Handling

Different errors encountered and dealt by us during the course of the project were:

- Assertion Error: When the assert statement failed, this error was raised.
- Attribute Error: This signal was raised when an attribute's assignment or reference went wrong.
- Import Error: When the position of a sequence was out of range, this value was raised.
- Name Error: When the variable wasn't found in either local or global scope, this flag was raised.
- Runtime Error: It was elevated because there was no other way to avoid making a mistake.
- Stop Iteration: The next() method sets this flag to signify that the iterator has completed returning items.
- Syntax Error: The parser displays this flag when a syntactic error occurs.
- Indentation Error: Raised when there was incorrect indentation.
- TabError: Indentation with uneven tabs and spaces caused this error to be raised.
- System Error: The flag is raised when the interpreter discovers an internal fault.
- System Exit: The sys.exit() procedure raised this value.
- Unbound Local Error: A mention of a local variable in a function is made, but still no data is associated with that variable, this exception is raised.

## CHAPTER 6

### Results, Conclusion, and Future Work

#### 6.1 Results from the experiments

Fig 15. below shows the accuracy of an individual model vs the accuracy gained by adding a pretrained model, where each model was trained from the ground up and fine-tuned on the Kinetics-600 database. We discovered that when pre-trained algorithms are used, the results show a significant improvement. It is observed that ResNet50's accuracy has increased by 30%, while ResNet101's efficiency has increased by 29% which is a substantial gain. The top 1 accuracy is the accuracy where the real class fits the model's most likely classifications, and it is the same as our standard accuracy. Top 2 accuracy refers to the accuracy with which the real class fits one of the model's two most likely classes. It also shows that when the model depth grows from Resnet18 to Resnet101, the accuracy of ResNet 3D rises by 12.5 percent, demonstrating that deeper Convolutional Neural networks fare better.

The highest accuracy was reached by Resnet 101 (84%) and Resnext 101(83%), whereas C3D had poor accuracy owing to the absence of pre-training. In contrast to ShuffleNetV2 and SqueezeNet, MobileNetV2 contains reversed residual blocks, which thrive at collecting dynamic movements, resulting in greater efficiency.

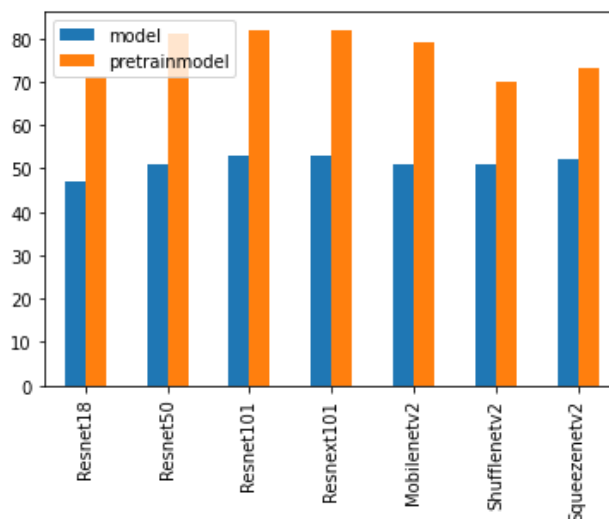


Figure 16. Comparative accuracy given by independent model and pretraining incorporated model

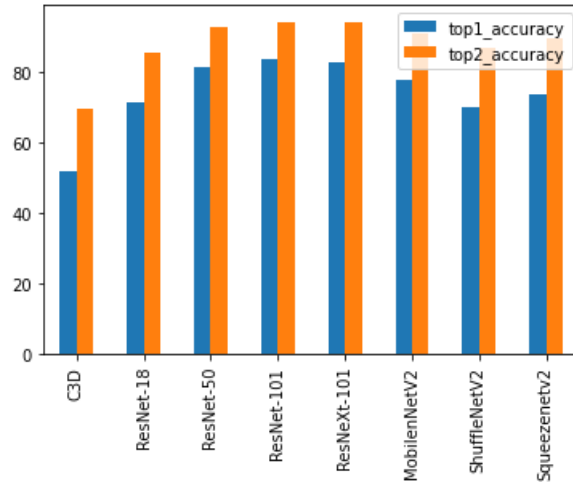


Figure 17. Top one and Top two Accuracies

Model	Top 1 Accuracy(%)	Top 2 Accuracy(%)
C3D	51.6	69.8
ResNet-18	71.3	85.4
ResNet-50	81.5	93.0
ResNet-101	84.3	94.5
ResNeXt-101	83.0	94.4
3D-MobileNetV2	77.8	90.8
3D-ShuffleNetV2	70.2	86.8
3D-SqueezeNet	73.6	89.6

Table 3. Table showing Top 1 and Top 2 accuracies of all models

## 6.2 Performance Evaluation Metrics

We used Precision, Recall and Confusion Matrix as performance evaluation metrics for our project. As previously stated, the neck pain and headache activities are quite similar, and as can be seen from the table below, the model mistakes the two and hence provides the least accurate results in both. Falling down, staggering, and fanning oneself, on the other hand, have unique characteristics, and the model seldom classifies them incorrectly. In situations like these, recall is crucial since the cost of a false negative is expensive and might result in the medical issue not

being recognised. In order to prevent ignoring an unpleasant health-related incident, having a low recall is also important. Falling down/staggering, which is more concerning, has a 99 percent recall rate, but headache has the lowest (64 percent).

Activity	Precision	Recall	F1-Score
Sneeze or Cough	0.79	0.76	0.78
Staggering	0.98	0.98	0.98
Falling down	0.99	0.99	0.99
Headache	0.70	0.64	0.67
Chest Pain	0.76	0.72	0.74
Back Pain	0.80	0.79	0.79
Neck pain	0.66	0.81	0.73
Nausea Vomiting	0.90	0.86	0.88
Fan Self	0.98	1.00	0.99

Table 4. Evaluation metric Scores - Resnext101 for detecting different actions

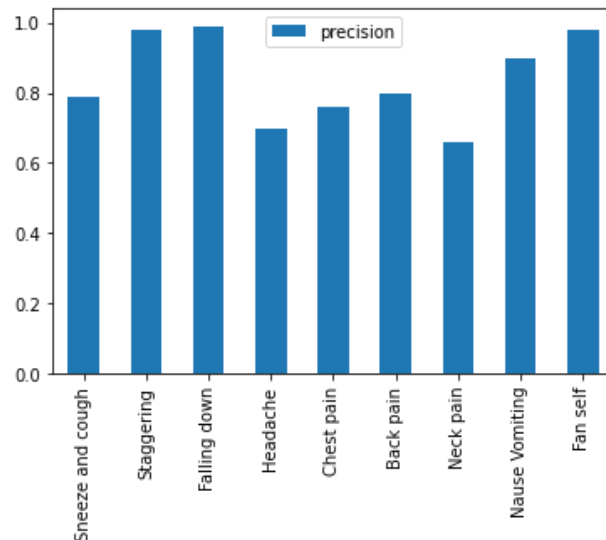


Figure 18. Precision score of Resnet101 model for detecting different action categories

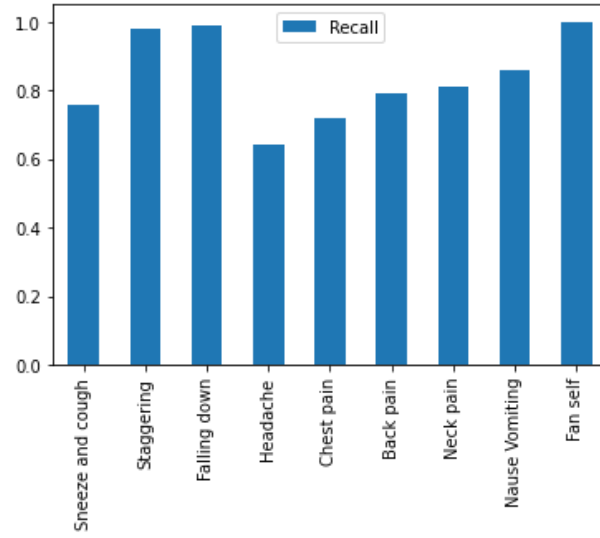


Figure 19. Recall score of Resnet101 model for detecting different action categories

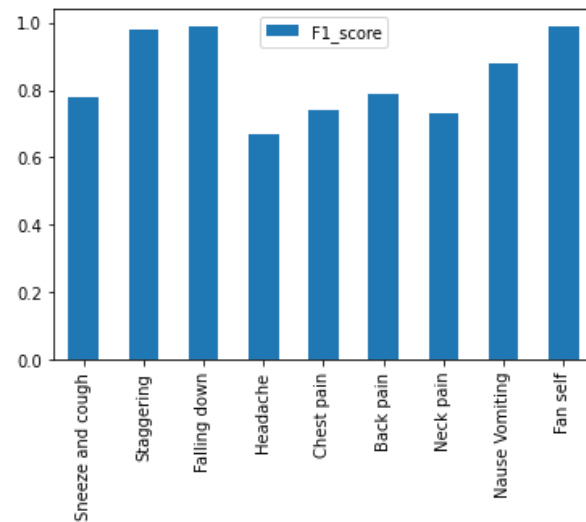


Figure 20. F1-score of Resnet101 model for detecting different action categories

Activity	Precision	Recall	F1-Score
Sneeze or Cough	0.77	0.77	0.77
Staggering	0.99	0.99	0.99
Falling down	1.00	1.00	1.00
Headache	0.69	0.67	0.68



Chest Pain	0.72	0.72	0.72
Back Pain	0.73	0.77	0.75
Neck pain	0.70	0.72	0.71
Nausea Vomiting	0.89	0.86	0.88
Fan Self	0.98	0.96	0.97

Table 5. Evaluation metric Scores - Resnext101 for detecting different actions

<b>Activity</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>
Sneeze or Cough	0.78	0.73	0.75
Staggering	0.99	0.99	0.99
Falling down	0.99	0.99	0.99
Headache	0.69	0.66	0.67
Chest Pain	0.68	0.69	0.69
Back Pain	0.74	0.75	0.75
Neck pain	0.65	0.71	0.68
Nausea Vomiting	0.85	0.84	0.84
Fan Self	0.98	0.98	0.98

Table 6. Evaluation metric Scores - Resnext101 for detecting different actions

## 6.2 Conclusion, Limitations and Future Work

The information gathered through activity monitoring may be utilized for a variety of purposes, including safe driving, crime rate management, and appropriate action, among others. In this project, we employed Activity Recognition to aid elder folks in making their lives easier and simpler. Since there is only a slight change in some of the activity videos, we planned to produce good results using newer deep learning models such as three dimensional CNNs, Three dimensional Resnet architectures (Resnet50, Resnet101, Resnext101) and three dimensional

Efficient 3D CNNs such as MobileNetV2, SqueezeNetV2 and ShuffleNetV2. Significant data preprocessing was done to extract maximum information possible from videos. It was observed that ResNet101 gave the best overall Top 1 accuracy of 84.3% and Top 2 accuracy of 94.5%. It also gave the best precision, recall and F1-Score in all action categories in comparison to other deep learning models. Falling down or staggering activity gave a 99 percent recall rate, but headache had the lowest score (64 percent). Also, we got the best Precision (0.99) as well as F1-Scores (0.99) while detecting falling down action. Our proposed solution will act as a healthcare monitoring system as keeping a trained caretaker may be costly and requires extra human effort. This will help us in detecting healthcare problems which affect elderly people beforehand. Current work in this area has a limitation that detection gets complicated when a person conducts many activities at the same time. Future study in this field might involve extracting additional features from videos utilizing real-time data rather than static datasets and systems with great processing capacity. To capture additional data, several sensors might be fitted while collecting data. Magnetic induction may also be used in conjunction with deep learning to recognize a wide spectrum of emotions. This might aid in improving our system's performance.

## References

- [1] Cho, S., Lim, S., Byun, H., Park, H., & Kwak, S. (2011). Human interaction recognition in YouTube videos. 2011 8th International Conference on Information, Communications & Signal Processing, 1-5.
- [2] Ke, S., Hoang, L.U., Lee, Y., Hwang, J., Yoo, J., & Choi, K. (2013). A Review on Video-Based Human Activity Recognition. *Comput.*, 2, 88-131.
- [3] Cheema, S., Eweiwi, A., & Bauckhage, C. (2012). Who is doing what? Simultaneous recognition of actions and actors. 2012 19th IEEE International Conference on Image Processing, 749-752.
- [4] Kong, Y., & Jia, Y. (2012). A Hierarchical Model for Human Interaction Recognition. 2012 IEEE International Conference on Multimedia and Expo, 1-6.
- [5] Kobayashi, I., & Noumi, M. (2010). A study on verbalizing human behaviors in a video. 2010 World Automation Congress, 1-6.
- [6] Ihaddadene, N., & Djeraba, C. (2008). Real-time crowd motion analysis. 2008 19th International Conference on Pattern Recognition, 1-4.
- [7] Singh, V.K., & Nevatia, R. (2011). Simultaneous tracking and action recognition for single actor human actions. *The Visual Computer*, 27, 1115-1123.
- [8] Ryoo, M.S. (2011). Human activity prediction: Early recognition of ongoing activities from streaming videos. 2011 International Conference on Computer Vision, 1036-1043.
- [9] Ke, S., Hoang, L.U., Lee, Y., Hwang, J., Yoo, J., & Choi, K. (2013). A Review on Video-Based Human Activity Recognition. *Comput.*, 2, 88-131.
- [10] Köpüklü, O., Kose, N., Gunduz, A., & Rigoll, G. (2019). Resource Efficient 3D Convolutional Neural Networks. 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), 1910-1919.
- [11] Hara, K., Kataoka, H., & Satoh, Y. (2018). Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet? 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 6546-6555.
- [12] Tran, D., Bourdev, L.D., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning Spatiotemporal Features with 3D Convolutional Networks. 2015 IEEE International Conference on Computer Vision (ICCV), 4489-4497.
- [13] Deotale, D., Verma, M., & P., S. (2021). Human Activity Recognition in Untrimmed Video using Deep Learning for Sports Domain. *Social Science Research Network*.
- [14] Ibrahim, M.J., Kainat, J., Alsalman, H., Ullah, S.S., Al-Hadhrani, S., & Hussain, S. (2022). An Effective Approach for Human Activity Classification Using Feature Fusion and Machine Learning Methods. *Applied Bionics and Biomechanics*, 2022.
- [15] Khurana, R., & Kushwaha, A.K. (2018). A Deep Survey on Human Activity Recognition in Video Surveillance. 2018 International Conference on Research in Intelligent and Computing in Engineering (RICE), 1-5.

- [16] Lin, W., Sun, M., Poovendran, R., & Zhang, Z. (2008). Human activity recognition for video surveillance. 2008 IEEE International Symposium on Circuits and Systems, 2737-2740.
- [17] Wendell H.(2020), "Activity Recognition with Still Images and Video,  
[http://cs230.stanford.edu/projects\\_winter\\_2020/reports/32636548.pdf](http://cs230.stanford.edu/projects_winter_2020/reports/32636548.pdf)
- [18] <https://rose1.ntu.edu.sg/dataset/actionRecognition/>