

# **SITTING POSTURE ANALYSIS USING CNN AND RCNN**

Submitted in partial fulfillment of the requirements for the award of  
Master of Technology degree in Medical Instrumentation

By

**NISHITHA R (Reg No. 40440005)**



**DEPARTMENT OF BIOMEDICAL ENGINEERING**

**SCHOOL OF BIO AND CHEMICAL ENGINEERING**

## **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade “A” by NAAC | 12B Status by UGC| Approved by AICTE**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

**MAY – 2022**



**SATHYABAMA**

INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE  
[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

**DEPARTMENT OF BIOMEDICAL ENGINEERING**

**BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **NISHITHA R (Reg No. 40440005)** who carried out the project entitled "**Sitting posture recognition using CNN AND RCNN**" under our supervision from **December 2021 to May 2022**.

**Internal Guide**

**Dr. T. SUDHAKAR, M.Sc., Ph.D.,**

**External Guide**

**Dr. N. R. KRISHNAMOORTHY, Ph.D.,**

**Head of the Department**

**Dr. T. SUDHAKAR, M.Sc., Ph.D.,**

---

**Submitted for Viva voce Examination held on 11-05-2022.**

**Internal Examiner**

**Dr. J. PREMKUMAR**

**External Examiner**

## DECLARATION

I, **NISHITHA R (Reg No. 40440005)** hereby declare that the Project Report entitled **“SITTING POSTURE ANALYSIS USING CNN AND RCNN”** done by me under the guidance of **Dr. T. SUDHAKAR** (Internal) and **Dr. N. R. KRISHNAMOORTHY** (External) at **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** is submitted in partial fulfillment of the requirements for the award of **Master of Technology** degree in **Medical Instrumentation**.

A handwritten signature in blue ink, appearing to read 'Nishitha R.', with a stylized flourish at the end.

**DATE:** 02-05-2022

**SIGNATURE OF THE CANDIDATE**

**PLACE:** Chennai

## ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **The Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sudhakar**, M.Sc., Ph.D., **Head of the Department, Department of Biomedical Engineering** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guides **Dr. T. Sudhakar**, M.Sc., Ph.D., and **Dr. N. R. Krishnamoorthy**, Ph.D., for their valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all **Teaching and Non-teaching staff** members of the **Department of Biomedical Engineering** who were helpful in many ways for the completion of the project.

## ABSTRACT

Due to pandemic curfew, work from home has been a lifestyle for past couple of year. Based on the statistics, the MSD (Musculo skeletal disorder) due to by improper work place have been raised from 1.4% (2019) to 37.7% (2021) and prolonged awkward posture increase 17.7 – 63.0% of neck pain and 23-34% of back pain. Though many workers use Ergonomic chair, complication of back pain and MSD raised due to improper posture in the chair. Major indication of improper posture is neck flexion and thoracic kyphosis. For the computer-based professionals and work from home employees, proper posture and hand movement are the important factor to be monitored in order to avoid the complication of MSD and CTS. This project aims to detect the angle of posture for the persons to indicate the improper positioning of body during work time. CNN and RCNN are used for the collection of various posture to identify and learn the erect position. Neural network is trained with available datasets. The live dataset from computer workers is fed to the neural network for analyzing the person's sitting posture and to indicate the correctness of the posture. Neural Network using Python is preferred to reduce the complexity in programming and to increase the accuracy of detection. Further work can be done by using other Neural network models and establish a comparative study on the two Neural network models and their accuracies.

**Keywords:** MSD- Musculo Skeletal Disorder; CTS- Carpel Tunnel Syndrome; CNN- Convolutional Neural Network, RCNN- Region based Convolutional Neural Network, Python

## TABLE OF CONTENT

| CHAPTER NO. | TITLE                                             | PAGE NO. |
|-------------|---------------------------------------------------|----------|
|             | <b>ABSTRACT</b>                                   | i        |
|             | <b>LIST OF FIGURES</b>                            | iv       |
|             | <b>LIST OF TABLES</b>                             | v        |
|             | <b>LIST OF ABBREVIATIONS</b>                      | vi-vii   |
| <b>01.</b>  | <b>INTRODUCTION</b>                               | 01       |
|             | 1.1 Spinal injury                                 | 01       |
|             | 1.2 Posture management                            | 02       |
|             | 1.3 Complication                                  | 02       |
|             | 1.4 Ergonomic workplace                           | 03       |
|             | 1.5 Neural Network                                | 04       |
|             | 1.6 Sitting posture analysis using neural network | 05       |
| <b>02.</b>  | <b>LITERATURE REVIEW</b>                          | 06       |
| <b>03.</b>  | <b>AIM AND SCOPE</b>                              | 10       |
|             | 3.1 Aim                                           | 10       |
|             | 3.2 Objective                                     | 10       |
|             | 3.3 Existing system                               | 10       |
|             | 3.4 Proposed system                               | 11       |
|             | 3.4.1 Advantages of proposed system               | 12       |
|             | 3.5 Scope and application                         | 12       |
| <b>04.</b>  | <b>MATERIALS AND METHOD</b>                       | 13       |
|             | 4.1 Hardware components                           | 13       |
|             | 4.1.1 Laptop                                      | 13       |
|             | 4.1.2 Camera                                      | 13       |
|             | 4.2 Software components                           | 13       |
|             | 4.2.1 Neural network                              | 13       |
|             | 4.2.1.1 Convolutional neural network              | 14       |

|                                                   |           |
|---------------------------------------------------|-----------|
| 4.2.1.2 Region based convolutional neural network | 14        |
| 4.2.2 Python software                             | 14        |
| 4.2.3 Keras                                       | 14        |
| 4.2.4 Gaussian filter                             | 14        |
| 4.2.5 OpenPose                                    | 15        |
| 4.2.6 TensorFlow                                  | 16        |
| 4.2.7 Torchvision                                 | 17        |
| 4.2.8 Coco- dataset for human                     | 17        |
| 4.3 Work flow                                     | 17        |
| 4.4 Methodology                                   | 18        |
| 4.4.1 CNN posture detection                       | 19        |
| 4.4.1.1 CNN Realtime posture detection            | 22        |
| 4.4.2 RCNN posture detection                      | 22        |
| 4.4.2.1 RCNN Realtime posture detection           | 23        |
| 4.5 Program coded                                 | 24        |
| <b>05. RESULT AND DISCUSSION</b>                  | <b>25</b> |
| 5.1 Output                                        | 25        |
| 5.2 Result                                        | 27        |
| 5.3 Discussion                                    | 30        |
| <b>06. SUMMARY AND DISCUSSION</b>                 | <b>32</b> |
| 6.1 Conclusion                                    | 32        |
| 6.2 Future scope                                  | 33        |
| <b>REFERENCES</b>                                 | <b>34</b> |
| <b>APPENDICES</b>                                 | <b>37</b> |

## LIST OF FIGURES

| FIGURE NO. | TITLE                         | PAGE NO. |
|------------|-------------------------------|----------|
| 1.1        | Stress in spine while sitting | 1        |
| 1.2        | Posture types                 | 2        |
| 1.3        | Poor posture complication     | 3        |
| 1.4        | Basic ergonomic workplace     | 3        |
| 1.5        | Neural network                | 4        |
| 4.1        | Keypoints                     | 15       |
| 4.2        | Workflow                      | 18       |
| 4.3        | Architecture of 3-Layer CNN   | 20       |
| 4.4        | Process of CNN                | 20       |
| 4.5        | RCNN Architecture             | 23       |

## LIST OF TABLES

| TABLE NO. | TITLE                    | PAGE NO. |
|-----------|--------------------------|----------|
| 4.1       | Modules and version used | 13       |
| 4.2       | Keypoints and ID         | 16       |
| 5.1       | Output of CNN            | 25       |
| 5.2       | Output of RCNN           | 26       |
| 5.3       | Result of CNN            | 28       |
| 5.4       | Result of RCNN           | 29       |

## LIST OF ABBREVIATIONS

| ABBREVIATION | FULL FORM                                           |
|--------------|-----------------------------------------------------|
| CNN          | Convolutional Neural Network                        |
| RCNN         | Region based Convolutional Neural Network           |
| FPN          | Feature Pyramid Network                             |
| MSD          | Musculo Skeletal Disorder                           |
| API          | Application Programming Interface                   |
| PCA          | Part Confidence Maps                                |
| COCO         | Common Objects In Context                           |
| PAF          | Part Affinity Field                                 |
| ANN          | Artificial Neural Network                           |
| CTS          | Carpel Tunnel Syndrome                              |
| FPGA         | Field Programmable Gate Array                       |
| VLSI         | Very Large-Scale Integration                        |
| ADC          | Analog to Digital Converter                         |
| SPS          | Sitting Pressure Sensor                             |
| ICAI         | International conference on Artificial Intelligence |
| IOT          | Internet Of Things                                  |
| FCN          | Fully Convolutional Neural Network                  |
| GAP          | Global Average Pooling                              |
| ML           | Machine Learning                                    |

|             |                            |
|-------------|----------------------------|
| DL          | Deep Learning              |
| GSR         | Galvanic Skin Response     |
| SVM         | Support Vector Machine     |
| FRS         | Force Resist Sensor        |
| LSTM        | Long short-term Memory     |
| AI          | Artificial Intelligent     |
| PC          | Personal Computer          |
| RAM         | Random Access Memory       |
| ReLU        | Rectified Linear Unit      |
| ROI Pooling | Region of Interest pooling |

# CHAPTER 1

## INTRODUCTION

### 1.1. SPINAL INJURY

The spine is a combination of bones, muscles, tendons and other tissues that reach from the base of the skull to the tailbone. The spine encloses the spinal cord and the fluid surrounding the spinal cord. The spine supports three natural curves: C-Shaped in the neck region, Reverse C-Shaped in the chest region and last curve in the hip region.

One of the major complications of the spine is Compression of the spine due to continuous sitting. Sitting for a prolonged period can be a major cause of back pain, cause increased stress on the back, arms, and legs and also can add a tremendous amount of pressure to the back muscles and spinal discs. Sitting in a slouched position can overstretch the ligaments and strain the spinal discs.

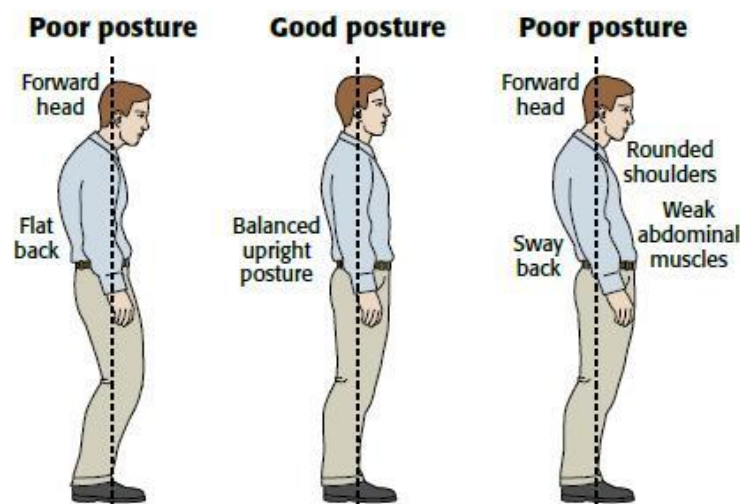
Poor posture and uncomfortable workplace ergonomics over time can damage the spinal structure and contribute to recurrent episodes of neck or back pain. Prolonged hunching leads to the back, core and abdomen muscles being strained and increasing the stiffness and weakness in the trunk and lower back. Herniation may result if the spine is bent forward for some time. The dynamics of the spine change when the spine is in a backward posture for a longer period.



***Fig 1.1 Stress in spine while sitting***

## 1.2. POSTURE MANAGEMENT

In a single day, every person undergoes many postures transformation like standing, sitting, stooping, lying down and many more, so learning proper posture management is important. Proper posture should be able to draw a straight line from the earlobe, shoulder, hip, knee, and into the middle of the ankle. Correct posture means keeping each part of the body in alignment with the neighbouring parts which will help in balanced and supported body structure. More than cosmetic appearance, good posture and back support are critical to reducing the incidence and levels of back pain and neck pain. Correct posture management will put the least amount of strain on the muscle and joints. Slouching, slumping, and other types of poor posture can cause muscle tension, back pain, joint pain and reduced blood circulation. Improper posture management may lead to breathing issues and fatigue. An unsupported sitting position places a small forward bend on the spine.



**Fig 1.2 Posture types**

## 1.3. COMPLICATION

Computer-based workers complain more frequently about the complication of the neck, back, shoulder and wrists. Most of the complications lie in Back pain, neck pain and shoulder pain. Poor posture may lead to the complication of tension and pressure introduced at joints and bones. These types of complications are regressive and progressive disorders that with less attention may lead to severe complications with irreversible deformation. Other than bone pain and deformations, poor circulation, poor digestion, constrained nerves, and change in curvature of the

spine may result. Improper posture for a longer duration may result in wear and tear of spinal discs. Poor posture complication includes increased injury risks, poor self-esteem, headaches, digestion problem, fatigue, premature ageing, disability and stress.



**Fig 1.3 Poor posture complication**

#### 1.4. ERGONOMIC WORKPLACE



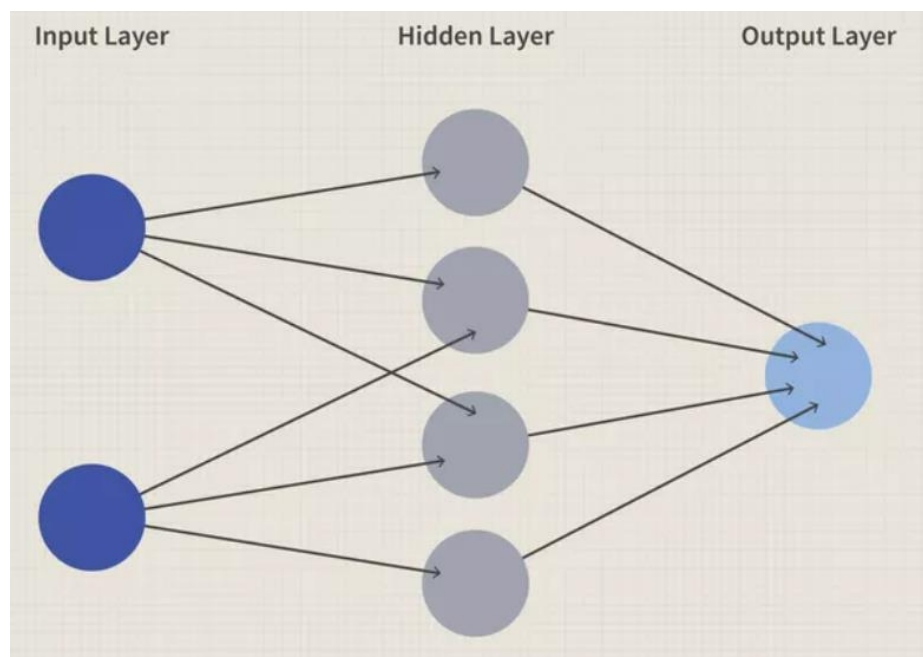
**Fig 1.4 Basic ergonomic workplace**

Workplace ergonomics is the science of designing the workplace, keeping in mind the capability and limitations of the worker. Poor workplace design leads to

fatigued, frustration and hurting workers. The design of the workplace aims to reduce strain, fatigue, and injuries by improving product design and workplace arrangement. The goal of the ergonomic workplace is a comfortable, relaxed posture. Computer ergonomics addresses ways to optimize computer workstations to reduce the specific risks of computer vision syndrome, neck and back pain and carpal tunnel syndrome. Researchers indicate that computer operators who use poorly designed chairs develop poor work posture and increased levels of musculoskeletal health symptoms.

### 1.5. NEURAL NETWORK

A neural network is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In general, neural networks refer to systems of neurons, either organic or artificial. The neural network can adapt to changing input so the network generates the best possible result without needing to redesign the output criteria. The neural network has 3 layers: An input layer for getting multiple inputs, a Hidden layer for having weights to internally fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs.



**Fig 1.5 Neural network**

## 1.6. SITTING POSTURE ANALYSIS USING NEURAL NETWORK

As many people adopt the work-from-home lifestyle the number of people affected by spine and neck disorders has increased to a greater extent. The primary cause of the complication of the disorder is that many professionals are not supported by the ergonomic workspace. All employees cannot make ergonomic workplaces which results in spinal complications. To prevent the chronicity of the disorder, preventive posture monitoring and sitting posture correcting tools have been emerging in the market. Both wearable and non-contact tools have been under research to make a preventive tool for position monitoring. Many sensors have been implemented in posture management like Force resist sensor which is placed on the cushion of sitting to calculate the force acting on the sensors, Pressure sensor to detect the evenly distributed pressure on the chair. Also, some research works on the non-contact supportive tools so that the persons are free from sensors. One of the predominant non-contact tools is the use of a Neural Network to monitor the posture of the person. Many research work has been carried out to extract accurate posture detection.

## CHAPTER 2

### LITERATURE REVIEW

Enrique Piñero-Fuentes, *et al.*, (2021). A Deep-Learning Based Posture Detection System for Preventing Telework-Related Musculoskeletal Disorders. MDPI Applied sciences. 21(15). An automated tool able to quantify the degree of incorrectness of a postural habit in a worker is needed. System based on the postural detection of the worker is designed, implemented and tested, using a specialized hardware system that processes video in real time through CNN. The tools used are Deep learning, cameras, Neural network classifier, skeletal processing mechanism, TRT pose. The evaluation of angle of rotation for each parameter and its classification are trained before with low power consumption. The drawback associated with the paper is that the left arm accuracy is low than other joint point's accuracy. The future works can be on real-time processing with an automatic recommendation system, a collection of incorrect poses during real work sessions to improve overall postures, changes to CNN can be used to get more detailed pattern recognition.

Sidrah Liaqat, *et al.*, (2021). A hybrid posture detection framework: Integrating machine learning and deep neural networks. IEEE SENSORS JOURNAL. 21(7). Novel hybrid approach based on machine learning classifiers (support vector machine (SVM), logistic regression (KNN), decision tree, Naive Bayes, random forest, Linear discrete analysis and Quadratic discrete analysis) and deep learning classifiers (1D convolutional neural network (1D CNN), 2D-convolutional neural network (2D-CNN), LSTM and bidirectional LSTM) to identify posture detection. The proposed hybrid approach uses prediction of machine learning (ML) and deep learning (DL) to improve the performance of ML and DL algorithms. Random forest model, SVM, KNN, smart cushion with AI, Hidden Markov model, GSR, TensorFlow are used. The analysis of different ML and DL techniques to extract with high precision in posture detection is obtained by this research paper. The drawbacks are that the use of combinational multiple algorithm technique may result in more computational time. The additional work for the neural network can be concentrated on increasing the speed of the processing algorithm.

Xu Ran, *et al.*, (2021). A portable sitting posture monitoring system based on a pressure sensor array and machine learning. Elsevier. 331. A portable sitting posture monitoring system to recognize the user's sitting posture and feedback the results in real time. A pressure sensor array is used to collect sitting postures related information, while the collected data can be displayed on a computer. The tools and modules used with the system are Raspberry pi, ANN, pressure sensor, vibrotactile feedback, vision sensor, LabVIEW. The monitoring system high accuracy, low power consumption with good human computer interaction. But the problem of inconvenient installation reduces the portability. The future works may rely on the compact design for ease portability and application of wearable sensors.

Ahmad, J. Sidén, J. and Andersson, H. (2021). A proposal of implementation of sitting posture monitoring system for wheelchair utilizing machine learning methods. MDPI. 21(19): 6349. A posture recognition system aimed at detecting sitting postures of a wheelchair user. The main goals of the proposed system are to identify and inform irregular and improper posture to prevent sitting-related health issues such as pressure ulcers. The monitoring system uses Raspberry Pi, SVM, kNN, RF, Decision tree, pressure sensor, PET sheets for detecting the sitting posture of the wheelchair person. The pros of this system are like accuracy due to the use of Neural Network is highly obtained. But this system has some drawbacks like calculation complexity, minimal training set results in lower accuracy, no apparent weight differences, longer compile time. The future works can be done on data collection improvement and data like demographical data for more privacy.

Gian Domenico Licciardo, *et al.*, (2021). A resource constrained neural network for the design of embedded human posture recognition systems. MDPI. 11(11). A custom HW design of a Fully Convolutional Neural Network (FCN) is presented in this paper to implement an embeddable Human Posture Recognition (HPR) system capable of very high accuracy both for laying and sitting posture recognition. The FCN exploits a new base-2 quantization scheme for weight and binarized activations to meet the optimal trade-off between low power dissipation, a very reduced set of instantiated physical resources and state-of-the-art accuracy to classify human postures. The tools used are Fully Convolutional Neural Network (FCN), Global average pooling (GAP), Keras. The Pressure sensors allow the design of more

compact systems which can be embedded in specific supports like chair and beds and able to capture small body deformation more accurately than cameras. It supports minimal power consumption and minimal hardware usage. But the computation time is high for the network and the use of sensors may provide discomfort to the persons. The further enhancement can be done by concentrating on improving the accuracy of the system.

Qisong Hu, Xiaochen Tang and Wei Tang (2020). A smart chair sitting posture recognition system using flex sensors and FPGA implemented artificial neural network. *IEEE sensors journal*. 20(14). The research article reports a novel Posture recognition system on an office chair that can categorize seven different health-related sitting postures. The system uses six flex sensors, an Analog to Digital Converter (ADC) board and a Machine Learning algorithm of a two-layer Artificial Neural Network (ANN) implemented on a Spartan-6 Field Programmable Gate Array (FPGA). This paper used Two-layer ANN implemented on FPGA with hyperbolic tangent function for VLSI implementation, less hardware overhead. This method is energy-efficient, private, and reliable, does not have a drift problem, user privacy, The major drawbacks associated with the paper are placement of sensors, battery backup, inaccuracy may occur due to sensor error, Placement of sensors, battery backup, inaccuracy may occur due to sensor error.

Yuanhang Feng, *et al.*, (2020). An analysis system of students' attendance state in classroom based on human posture recognition. *19th International Symposium on Distributed Computing and Applications for Business Engineering and Science*. 1, pp. 214-217. The recognition of the human body key points, including the rather important points of one's body. There are many efforts to be made before the system could fit the requirement of the specific environment, such as semantic segmentation and super resolution. The system would also have clear visualization when finally faces customers. The tools used are mask- RCNN, Microsoft Kinect, semantic based image segmentation and surveillance cameras. The system has strong portability, semantic based algorithm produced high accuracy. But the person under the video is often blurred and it needs for super resolution and more test subjects are required.

Jahnvi Gupta, *et al.*, (2020). An introduction to IoT based posture detection and

underlying sensor technology. ResearchGate. Internet of Things (IoT) based solution are being planned and used for the human posture recognition and detection. The data collected from sensors is then passed to machine learning or deep learning algorithms to find different patterns. The tools used for the posture detection are IOT, falls sensor, KNN, PCA and Arduino. This research has more accuracy for the system and the drawback of the system is the sensor drift, multiple sensors may affect the precision. The future scope of this paper is the selection of transmission interval and data collection that for multiple sensors is affected. Sensor drift is another challenging problem with time- sensitivity of sensor insensate.

Arif Reza Anwary, *et al.*, (2021). Smart-cover: a real time sitting posture monitoring system. Elsevier, Sensors and Actuators. 317. A Smart-Cover, an automatic system to provide real time visualization and information about sitting posture. A Sitting Pressure Sensor (SPS) is built using Velostat, conductive fabric and foam to collect pressure distribution information using IoT and rule-based classifier. This can be obtained by using IoT, Fuzzy rule-based classifier, Android application, SPS sensor. The advantage of the system is easy visualization and monitoring high reliability. Drawbacks of the system are pressure sensors limits the mobility and it provides discomfort to the persons. Future scope of the method is to aim for enhancing the reliability and validity of monitoring the posture.

Rabia Hasib, *et al.*, (2021, April 07). Vision-based human posture classification and fall detection using convolutional neural network. International conference on Artificial Intelligence (ICAI). A novel computer vision-based system for detecting falls on the basis of a monocular camera. The recorded video data, mask regional convolutional neural network (mask R-CNN) is first applied to detect and extract human silhouettes. This method is executed by using Mask RCNN, CNN, Computer vision, Python, Colab. This paper has the advantage Feasible, easy to use with zero electromagnetic interference, Mask RCNN generate high quality masks. This paper does not have much accuracy in feature extraction and classification accuracy is low. The future scope of this research is to extend the study under night environment and study under indoor and outdoor environment.

## CHAPTER 3

### AIM AND SCOPE

#### 3.1 AIM

This proposal aims to design software using the Neural network for monitoring the sitting position of the person which is a contactless tool for providing comfort and high accuracy. The research aims to have a comparative study in two Neural Networks based on their latency in producing the output for each input and to provide healthy posture management by reducing the chronic complication of improper positioning.

#### 3.2 OBJECTIVE

To develop code for both Convolutional Neural networks and Region-based convolutional neural networks.

To feed datasets for training the system.

To analyse the response of CNN and RCNN for the given image and live samples.

To analyse and compute the time latency given by both CNN and RCNN.

To classify major 3 postures of the person: Hunchback, Straight and Reclined.

#### 3.3 EXISTING SYSTEM

Research for sitting posture detection and analysis has been increasing for the past couple of years. Predominantly posture detection is carried out either by wearable sensors or by Machine learning methods. Many research works stated the use of sensors like Flex sensors, flexible array pressure sensors, FRS, and Sitting pressure sensors (SPS) will help in measuring the pressure and/or force exerted over the cushion while sitting. Some articles state the combinational use of sensors and Neural networks for obtaining more accuracies. Technologies like IoT are used alone with sensors and machine learning for storing the data in the cloud. By using sensors, the changes in pressure and/or force are monitored so that the persons are alerted when the values exceed the threshold in any of the sensors being used. To remove the noise and artifacts during sensor recording, they are supported by various filters like the Kalman filter. The monitoring and value analysis are made easy by creating a supporting android application. Some research uses k-Nearest

neighbours, support vector machine, random forest, and decision tree for experimental analysis along with raspberry pi. A fully conventional neural network to implement the embeddable human posture recognition system is used and proved that the accuracy is achieved at 98.99%. The use of Kinect V2 for the classification of human body joints is like using the OpenPose library. For performing the image classification either MATLAB or python are used. In a research paper, integration of SVM, CNN, and LSTM are made to study the posture of a person. 3-D RCNN with the spatiotemporal process is used for the multi-stage technique to recognize the position of a person. The neural network is emerging in this domain as they produce a high range of accuracy with greater classification. The optical neural network has a random shift wavelet pattern for easy posture recognition.

### 3.4 PROPOSED SYSTEM

The sitting posture analysis using CNN and RCNN is a comparative work to analyse the accuracy and latency of the CNN and RCNN in detecting the posture of the person. The output of the network indicated the keypoints extracted from the given input and the latency for detection of the system. The postures that we considered to detect the position are: a. Hunchback, b. Reclined, c. Straight, d. hand folds, e. kneeling, f. cross legs. This software detects the sitting position by getting the input from the lateral view of the person as we aim to calculate the posture by calculating the angle and degree between the keypoint of reference. The keypoints are extracted from the image by using the OpenPose model. In this project, 18 keypoints in the body are extracted and analysed.

Initially, the software gets the input image and extracts the keypoints from the person. This is achieved by constructing three python files: one for containing the whole architecture of the software, second containing the parameters essential to predict the keypoints, third having the functions to calculate the coordinates of keypoints.

Once the keypoints are extracted the system starts detecting the angle between the keypoints to analyse the deviation of the spine from the straight posture. Here we considered Hip and Ear keypoint since both the keypoints lay on the same line when they are in a straight position. This project has four programs:

- a. Posture detection in an input image using CNN
- b. Posture detection in an input image using RCNN
- c. Posture detection in an input video and real-time video input using CNN
- d. Posture detection in an input video and real-time video input using RCNN

#### **3.4.1 Advantages of the proposed system:**

Non-contact device to monitor the position cost-efficient tool since no more hardware is involved in the development. Since the program produces good accuracy with a normal web camera and default camera in PC and Laptop respectively, the hardware used is minimized. This research paper proposes a comparative study over CNN and RCNN, it helps in using an effective type of neural network.

### **3.5 SCOPE AND APPLICATION**

A non-contact neural network tool for making an analytical study in a sitting position is highly helpful to monitor the computer workers and other sitting related workers to avoid progressive growth of the spine and neck-related complications as a result of poor posture management. By giving a comparative study on types of neural networks, it would help categorize the neural networks based on their functionalities. The live video monitoring and checking the posture often will help the person to alert the posture change. Moving towards the AI, ML and Neural network is emerging as the accuracy is obtained at a high range and the complexity is not more when compared with the hardware-based system. This method has a broad range of applications since the method is a contact-less tool.

## CHAPTER 4

### MATERIALS AND METHODS

#### 4.1 HARDWARE COMPONENTS

##### 4.1.1 Laptop

The program used for testing the sitting posture is coded for testing the input in a real-time environment. Please sit conveniently at the proper lateral view for detecting the live input at high accuracy. A system with at least 8GB of RAM is required to run the code on a laptop. If a 4 GB RAM supported system is used then the output may have rough output than smooth output with output lag from the input frame.

##### 4.1.2 Camera

If the laptop does not support a camera or if the user is using a PC, then an external web camera with normal resolution can be supported for detecting the input image or video signals.

#### 4.2 SOFTWARE COMPONENTS

**Table 4.1 Modules and versions used**

| S.NO | MODULE                | VERSION  |
|------|-----------------------|----------|
| 1    | NumPy                 | 1.15.4   |
| 2    | Keras                 | 2.2.4    |
| 3    | Keras-Application     | 1.0.6    |
| 4    | Keras-Preprocessing   | 1.0.5    |
| 5    | Scipy                 | 1.0.0    |
| 6    | OpenCV-contrib-python | 3.4.2.16 |
| 7    | OpenCV-python         | 3.4.2.16 |
| 8    | TensorFlow            | 2.8.0    |
| 9    | Torchvision           | 0.11.3   |

##### 4.2.1 Neural Network

Neural Network is a part of Machine Learning which makes the device learn

itself with the existing inputs. As the name implies, a Neural network resembles the structure of Brain Neurons having many networks of weight layers such that the accuracy and the learning rate of the device are being improvised.

#### *4.2.1.1. Convolutional Neural Network*

The Convolutional Neural Network is a part of Neural Networks that performs the mathematical convolution to train the network. This project uses a three-layer CNN. The CNN has a feature map that generates both Part Confidence Maps and Part Affinity Field.

#### *4.2.1.2. Region based Convolutional Neural Network*

The region-based convolutional neural network is a more derived version of a neural network that concentrates deeply on the region-based classification of the object in the image. RCNN is used because it mainly concentrates on image classification based on region detection and object segmentation. Dataset is supported by the MS-COCO dataset. Keypoint RCNN is slightly different from Mask-RCNN.

#### **4.2.2 Python Software**

For coding and training the Neural networks, python is being used as python is one of the general-purpose, user-friendly programming languages and is easily adopted by many non-programmers. Python along with libraries like TensorFlow, Scikit-learn, Keras and Pytorch can be used for Machine learning tasks.

#### **4.2.3 Keras**

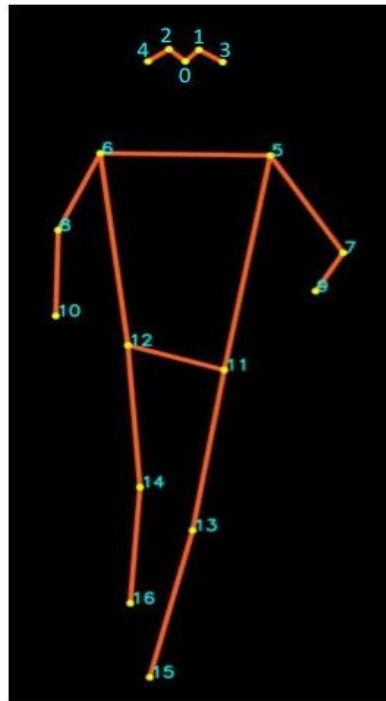
Python wrote a deep learning API that runs on top of the machine learning platform TensorFlow. This API is simple, flexible, and powerful and used for research as it supports the fast experimental procedure. Layers and models are the core data structures of Keras. Boundary Box regression is carried out in Keras for object detection and identification.

#### **4.2.4 Gaussian filter**

The Gaussian filter is a low-pass filter. This filter is used for reducing the noise and blurring in an input image. Here the multidimensional Gaussian filter is used. It targets to retain the edges so that keypoint extraction is made at ease. This filter helps in maintaining the smooth texture of the image. To extract fine details in input image or video by reducing the noise in the input is done by this Gaussian blur.

#### 4.2.5 OpenPose

OpenPose is a multi-person detecting real-time library that can be used for detecting the human body's keypoints. OpenPose is written in C++ and Caffe. OpenPose with python can support the creation of different operating systems and languages. Open pose library is used for body parts keypoint detection using CNN. This library has the capability of jointly detecting the human body, face and foot keypoints. These keypoints coordinates can be used to estimate the sitting posture of the person. By iterating the detection part of the code, a set of keypoints can be detected for multiple person input. Since this proposal concentrates on single person input, the iterating procedure is not carried out in Open Pose. This Open Pose is used along with the Keras model for extracting the keypoint features from the input. OpenPose is used to extract the sitting posture characteristics with Keras Deep learning framework to build a deep learning network, and use the dataset to train the network. Attitude estimation like human motion, facial expression, and finger movement can be achieved. Open pose has excellent robustness which provides a bottom-up approach to the real-time estimation of multi-person gestures without the need for any character detectors. The Open Pose supports 18 body joints and 17 line connecting joints.



**Fig 4.1 Keypoints**

**Table 4.2 Keypoints and ID**

| ID | PART           |
|----|----------------|
| 0  | Nose           |
| 1  | Left eye       |
| 2  | Right eye      |
| 3  | Left ear       |
| 4  | Right ear      |
| 5  | Left shoulder  |
| 6  | Right shoulder |
| 7  | Left elbow     |
| 8  | Right elbow    |
| 9  | Left wrist     |
| 10 | Right Wrist    |
| 11 | Left hip       |
| 12 | Right hip      |
| 13 | Left knee      |
| 14 | Right knee     |
| 15 | Left ankle     |
| 16 | Right ankle    |

#### **4.2.6 TensorFlow**

An open-source platform for machine learning where the pose estimation can be achieved by merely estimating the key points and key joints output. It is a core library and has a comprehensive, ecosystem of tools, and community resources that help the programmer to have state of art in the development of the Neural network. Pose estimation refers to computer vision techniques that detect human figures in images and videos. The pose estimation merely estimates where key body joints are and do not recognize who is in an image or video. The keypoints detected are indexed by a

part ID, with a confidence score between 0.0 and 1.0. The confidence score indicates the probability that a keypoint exists in that position. Tensor flow lite pose estimating models: Move Net, Pose Net.

#### **4.2.7 Torchvision**

Torchvision is a model that contains a pre-trained key point detection model which is built on top of the ResNet-50 FPN. FPN is a process of fusing features maps at multiple scales for preserving the information at multiple levels. Torchvision contains the definition of models for addressing different tasks, including image classification, pixel-wise semantic segmentation, object detection, instance segmentation, person keypoint detection, video classification and optical flow. The pre-trained models for detection, image segmentation and keypoint detection are initialized with the classification models in Torchvision.

***torchvision.models.detection.keypointrcnn\_resnet50\_fpn***

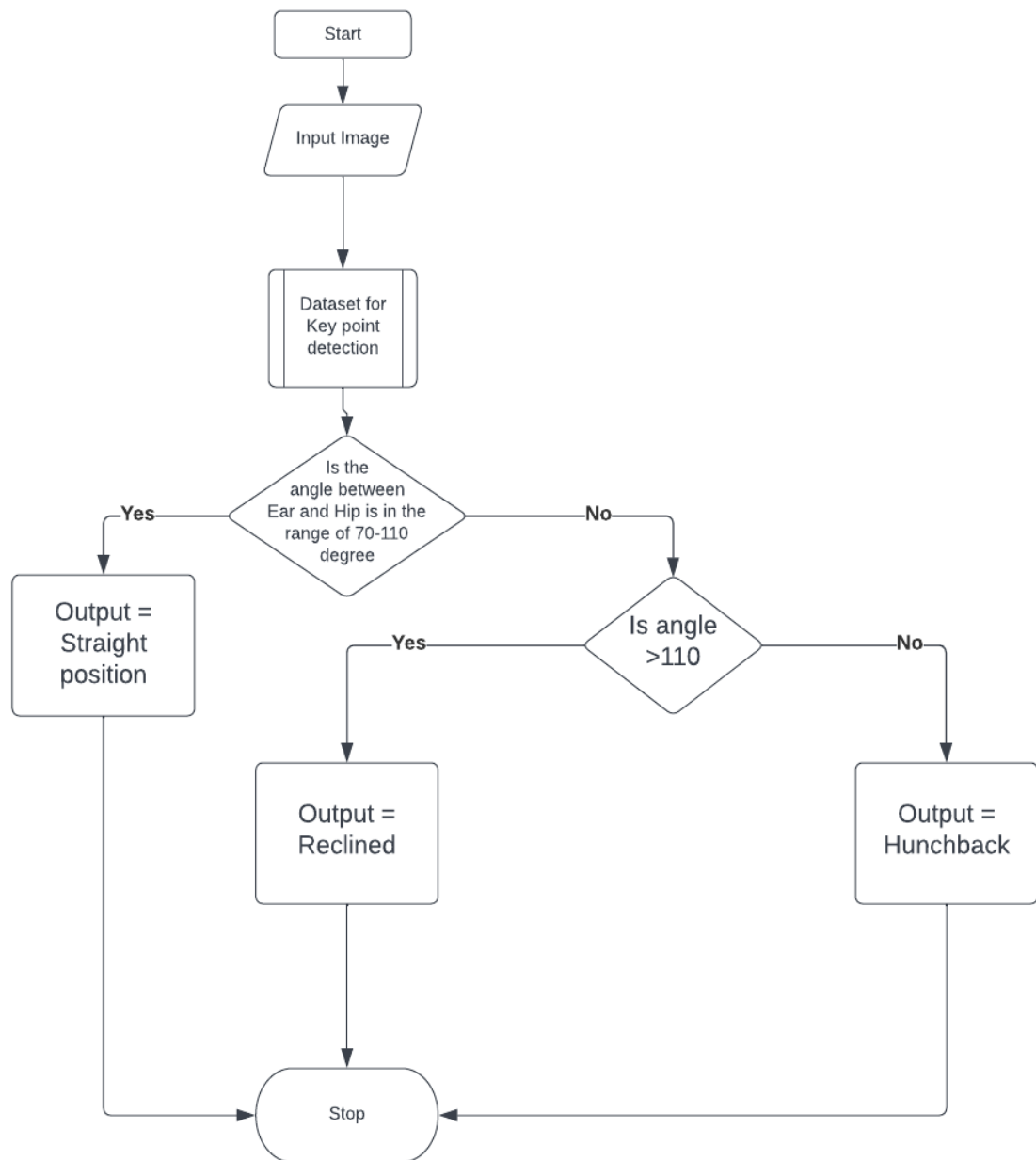
#### **4.2.8 COCO- Dataset for Human**

COCO is object detection and segmentation on large scale. COCO has features like object segmentation, recognition, pixel segmentation, 250,000 people images with keypoints from 350,000 images. The main 16 keypoints in human skeleton detection are taken to train the Network. MS COCO dataset can be obtained by the official website with a python script to store object detection of the COCO dataset.

### **4.3 WORK FLOW**

Steps carried to recognize the sitting posture are:

1. Gather the keypoint coordinates for different body parts
2. Calculate the sitting angle of the person
3. Getting the ear position
4. Getting the hip position
5. Calculating the angle between ear and hip
  - a. If angle > 110 → Reclined position
  - b. Elif angle > 70 → Hunchback position
  - c. Else → Straight position



**Fig 4.2 Workflow**

#### 4.4 METHODOLOGY

Initially the project phase is started with testing the Neural Network with google dataset for proper object estimation. The accuracy of the networks is observed and then the system is trained with COCO dataset and OpenPose keypoint detection techniques. Then the real time dataset is created and fed into the CNN and RCNN system to check for the accuracy. The posture detection is calculated by degree of

deviation between Ear and Hip position key points. Then RCNN and CNN is tested for live stream video sample to extract the time consumed to produce the result. The last phase of the project is carried out by comparing the two Neural Network output and the time taken by the system to produce the output. This proposal uses CNN and RCNN for analysis of sitting position and also aims in producing a comparative study between CNN and RCNN. Torchvision is used for pretrained keypoint detection which is built on the top of ResNet-50 FPN backbone. For the keypoint extraction Open pose library and human joint keypoints from COCO datasets are used. The project uses Keras deep learning API which runs on top of TensorFlow platform for fast experimental procedure and used for the pose estimation. The noise in the input image and video data can be eliminated by use of multidimensional Gaussian noise for retaining the edge in the image for keypoint extraction.

#### **4.4.1 CNN Posture detection**

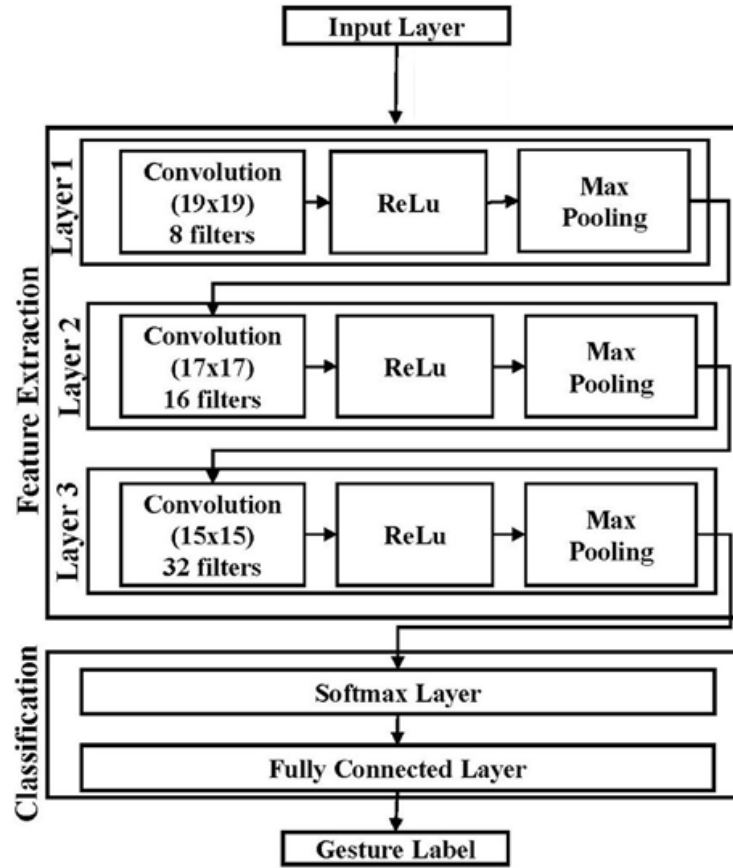
Initially, the image is passed through the baseline CNN network to extract the feature maps of input. The feature map is processed in 3 stages CNN to obtain are Part Confidence Maps and Part Affinity Field. Greedy bipartite matching algorithms are used to get PCA and PAF.

Confidence Map is a 2D representation that a particular body part can be located in pixels. PAF is a set of 2D vector fields that encodes the location and orientation of limbs of people. It encodes pairwise connections between body parts.

Three steps carried out in CNN are:

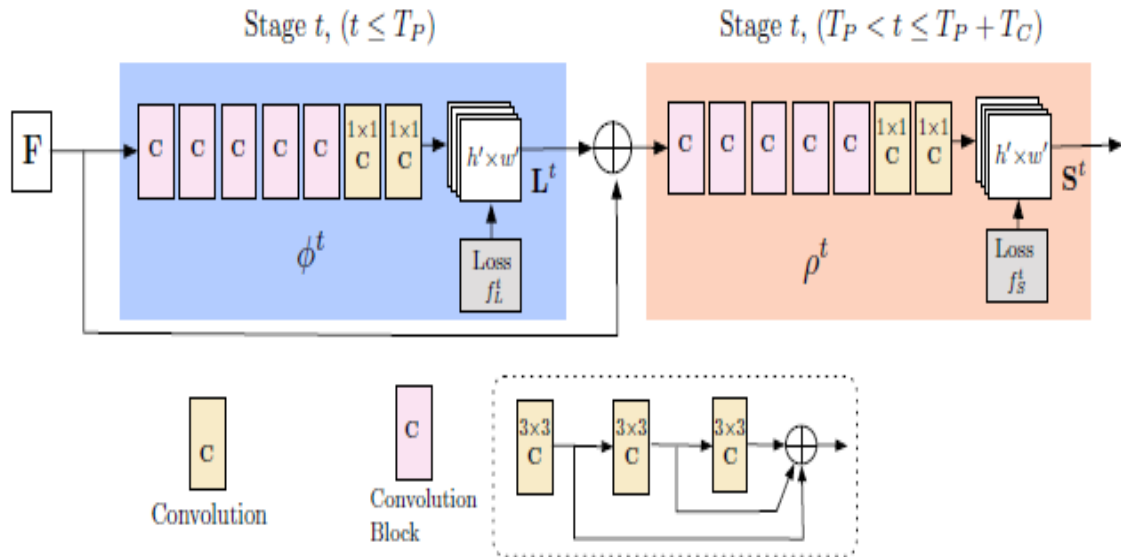
1. To predict the Part Affinity Fields from the Feature maps of the base network
2. Use the output PAF from previous layers to refine the prediction of Confidence maps
3. Confidence map and Part Affinity Field are passed into the greedy algorithm

The loss function is used to calculate the loss between PCM and PAF to the ground truth maps and fields.



**Fig 4.3 Architecture of 3-layer CNN**

The layers of CNN are carried out with 3 layers of process with use of PAF and PCM with ReLU and Maxpooling methods. The major 3 process with loss function are explained using the following figure.



**Fig 4.4 Process of CNN**

Confidence maps:

$S = (S_1, S_2, \dots S_J)$  WHERE  $S_j \in \mathbb{R}^{w \times h}$ ,  $j \in 1 \dots J$ ;  $J$  is number of body parts location

A Gaussian curve with gradual changes where *sigma* controls the spread of the peak. The predicted peak of the network is an aggregation of the individual confidence maps by a max operator.

$$S_{j,k}^*(p) = \exp\left(\frac{-||p, x_{j,k}||^2}{\sigma^2}\right)$$

Part Affinity fields:

The part affinity field is required especially in multi person pose detection we are required to map the correct body parts to its body. Because for multiple persons, there are multiple heads, hands, shoulders etc. Thus, it becomes difficult to distinguish sometimes when they closely grouped together. PAF provides a connection between different part of the body that belongs to the same person. A stronger PAF link between body parts represents those high chances that those body parts belong to the same person

$$L = (L_1, L_2, \dots L_C) \text{ WHERE } L_c \in \mathbb{R}^{w \times h \times c}, c \in 1 \dots C$$

First stage of CNN:

This stage predicts the Part Affinity Fields refines  $L_t$  from the feature maps of base network  $F$ .

$$L_t = \Phi^t(F, L^{t-1}), \forall t \leq T_p,$$

Second stage of CNN:

This stage takes use the output Part Affinity Fields from the previous layers to refine the prediction of confidence maps detection.

$$S^{T_p} = \rho^t(F, L^{T_p}), \forall t = T_p,$$

The final  $S$  (confidence maps) and  $L$  (Part Affinity Field) are then passed into the greedy algorithm for further process.

$$S^t = \rho^t(F, L^{T_p}, S^{t-1}), \forall T_p < t \leq T_p + T_c$$

Loss Function:

An *L2-loss* function is used to calculate the loss between the predicted confidence maps and Part Affinity fields to the ground truth maps and fields.

$$f_L^{ti} = \sum_{c=1}^C \sum_p W(p) \cdot \|L_c^{ti}(p) - L_c^*(p)\|_2^2$$

$$f_S^{tk} = \sum_{j=1}^J \sum_p W(p) \cdot \|S_j^{tk}(p) - S_j^*(p)\|_2^2$$

where  $L_c^*$  is the ground truth part affinity fields,  $S_j^*$  is the ground truth part confidence map, and  $W$  is a binary mask with  $W(p) = 0$  when the annotation is missing at the pixel  $p$ . This is to prevent the extra loss that can be generated by these masks. The intermediate supervision at each stage is used to address the problem of vanishing gradient problem by replenishing the gradient periodically.

The replenishing gradient:

$$f = \sum_{t=1}^{Tp} f_L^t + \sum_{t=Tp+1}^{Tp+Tc} f_S^t$$

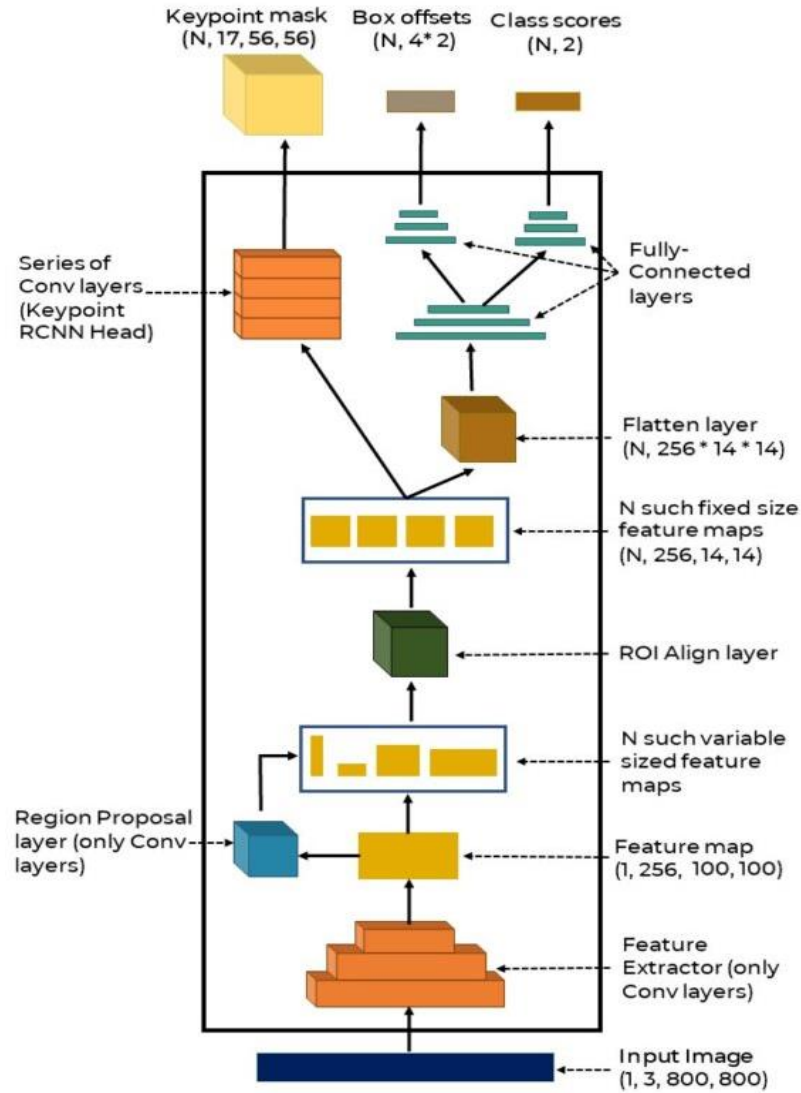
#### 4.4.1.1 CNN Realtime Posture detection

The CNN is coded for detecting the live input video streaming on the web camera attached to the laptop/PC and to process the recorded video input for detecting the posture of the person. The Torchvision is used for the dataset with a gaussian filter to remove the noises in the input. If the posture is not able to detect or if the camera is not in the lateral position to the person, then the system responds with a notification of “unable to calculate the angle”.

#### 4.4.2 RCNN Realtime Posture detection

Keypoint RCNN has the same functionality as Mask-RCNN, they differ by output size and the way keypoints encode in the keypoint mask. Keypoint RCNN has one-hot encoding a keypoint of the detected object instead of the whole mask. Class-wise output feature map from Mask-RCNN: Here the feature map has two channels person and the background class. Pretrained keypoint detection model built on top of the ResNet– 50 FPN backbone. FPN is a fusion feature map at different scales so that the information is preserved. Faster RCNN has numerous layers where the Region-proposal- layer predicts the locations of  $N$  number of objects in feature maps. These regions are individually passed to the ROI-Pooling layer which resizes the feature maps by quantising the size grid to a fixed-size grid and getting the max-values to place in the fixed grid. A Fully-Connected (FC) layer follows the ROI-pooling layer. Mask-RCNN has some modifications of layers, like the ROI-Align

layer is used instead of the ROI-Pooling layer and the output of ROI-Align is passed to the Mask-RCNN head. COCO dataset has 80 classes for detection and segmentation, the annotations are offered for person class. Torchvision is particularly trained to identify key points in a person.



## KEYPOINT RCNN

**Fig 4.5 RCNN Architecture**

### 4.4.2.1 RCNN Realtime Posture detection

RCNN is coded for analysing the live streaming video and recorded video input. The real-time input is taken and processed with the dataset from Torchvision and the noise is removed by using the Gaussian filter. If the camera is in the wrong position or if the lateral view of the subject is not properly detected, then the system

comes with an alert message of “Unable to calculate the angle”, later which the position can be altered.

#### 4.5 PROGRAMS CODED

- Model.py- Contains architecture of the model.
- Config\_reader.py- Contains the parameters that are essential for the model to predict the key points. Keeping the specifications of the system in mind.
- util.py- Some functions required to calculate the co-ordinates of the key points.
- CNN posture detection.py – Contains code for CNN based image posture detection.
- CNN real time posture detection.py – Contains code for CNN based live video posture monitoring.
- RCNN posture detection.py - Contains code for RCNN based image posture detection.
- RCNN real time posture detection.py - Contains code for CNN based live video posture monitoring.

## CHAPTER 5

### RESULT AND DISCUSSION

#### 5.1 OUTPUT

This project is to perform an analysis of two different Neural networks used for the recognition of human sitting posture. The outcome of this analysis is numerical values which concentrate on the degree of deviation and the time taken by the neural network to calculate the angle of deviation between the Ear and Hip keypoint. Based on the analysis, both RCNN and CNN produce correct posture detection and the angle of deviation is also similar and ranges on the same scale with a fluctuation of +2 degrees. The only difference in RCNN and CNN output making RCNN more predominant than CNN is the latency of the two neural networks. Time consumed by RCNN is much lower than CNN for calculating position and degree of deviation in the given input. As the keypoint of reference for calculating the position are Ear and Hip, the camera has to be kept in the lateral view for analyzing the position of the person.

The comparison of CNN and RCNN output time for the same set of images is represented below:

**Table 5.1 Output of CNN**

| <b>Input (Image)</b> | <b>Expected Result (Position)</b> | <b>Actual Result (Position)</b> | <b>Time in Seconds</b> |
|----------------------|-----------------------------------|---------------------------------|------------------------|
| Sample 1             | Hunchback                         | Hunchback                       | 56.30753               |
| Sample 2             | Hunchback                         | Hunchback                       | 61.99616               |
| Sample 3             | Hunchback                         | Hunchback                       | 72.53529               |
| Sample 4             | Reclined                          | Reclined                        | 52.14701               |
| Sample 5             | Reclined                          | Reclined                        | 70.73289               |
| Sample 6             | Reclined                          | Reclined                        | 64.77853               |
| Sample 7             | Straight                          | Straight                        | 82.54892               |
| Sample 8             | Straight                          | Straight                        | 69.10063               |
| Sample 9             | Straight                          | Straight                        | 69.49605               |

Based on the output of CNN, it is clear that the time taken for CNN for identifying a position is around a minute or more than a minute. Though this latency is considerable, this network performs at low when it is implemented for real-time live capturing of video data. One way to minimize the latency of the system is to increase weight layers in the network and to train the network with more datasets

**Table 5.2 Output of RCNN**

| <b>Input (Image)</b> | <b>Expected Result (Position)</b> | <b>Actual Result (Position)</b> | <b>Time in Seconds</b> |
|----------------------|-----------------------------------|---------------------------------|------------------------|
| Sample 1             | Hunchback                         | Hunchback                       | 9.21453                |
| Sample 2             | Hunchback                         | Hunchback                       | 9.80878                |
| Sample 3             | Hunchback                         | Hunchback                       | 9.56103                |
| Sample 4             | Reclined                          | Reclined                        | 8.6486                 |
| Sample 5             | Reclined                          | Reclined                        | 8.51954                |
| Sample 6             | Reclined                          | Reclined                        | 8.89083                |
| Sample 7             | Straight                          | Straight                        | 8.1843                 |
| Sample 8             | Straight                          | Straight                        | 8.41329                |
| Sample 9             | Straight                          | Straight                        | 9.71633                |

Based on the output of RCNN, it is clear that the latency of this neural network is very less when compared to CNN. The Time taken to calculate the posture of input data is less than half a minute. This latency is much more suitable for real-time input processing as output can be extracted. In comparing both neural networks, the accuracy of position detection is similar and the calculation of the degree of deviation between keypoint is similar. Based on the time consumed by both neural networks, the priority shifts towards RCNN. Since RCNN is a modified mask RCNN which is a modified version of fast CNN, time variation between CNN and RCNN is seen at the larger interval.

The output of real-time RCNN and CNN code are extracted with running commands of position notification. The live input is given by placing the laptop at the side of the

person so that a lateral view of a person is taken. Even when the camera is placed at the front side output can be extracted but for proper keypoint detection, the system has to be placed at the lateral view covering the position of the Hip and ear of the input person data.

## 5.2 RESULT

The primary objective of this analysis paper is to study the position of a person while using the computer or while sitting so that the chronic complication of the spine deformity and back pain can be reduced or prevented. Based on the study, proper posture management has reduced spinal disorders and prevented deformities to a greater extent. As aimed, this paper developed two Neural Networks namely CNN and RCNN code using Python language. Python is preferred as this makes code understandable and also makes the neural network trained with the existing COCO dataset. Both the code for RCNN and CNN are trained with available COCO datasets and convolution is performed in the Model.py file. The human skeletal keypoint extraction is used for detecting the keypoints using Keras and TensorFlow. Commonly used keypoints are 16 in which we concentrate on the two keypoints: Ear and Hip. Since the Ear axis and Hip axis of the human body are in-phase and a straight line, these two points are used in this analysis paper to find the angle of deviation.

Obtaining and analyzing the live streaming video of a person is also targeted in this research paper. So, code for RCNN and CNN live video real-time input is also coded and executed. This code has the output as live text being printed for every change in input video data. Different posture in different environment of computer working person's sitting posture images has been given as test input for the analysis of the CNN and RCNN precision and to calculate how fast the networks get trained. For comparing, the same set of input images has been given to both the Neural network and the output keypoint extracted images with calculated time and degree of deviation between Ear and Hip keypoint is taken for analytical study. And based on the result, it is clear that the CNN has more latency period in detecting the posture than the time taken by the RCNN network. This is because the computational time taken by RCNN is less when compared with CNN. Since the keypoint RCNN is a derived form of Fast RCNN, time image segmentation, and object classification are

reduced to a greater extent when compared to normal CNN-based image classification and image feature extraction. Real-time RCNN and CNN have dual displays, one displaying the live video input with keypoint detection marks and another screen with running commands displaying the live position of the input person and alert message to change or to stay in the current posture. If the image/video is not recognized by the network, then the system provides an alert message that the input is not recognized.

Tabular column 5.3 and tabular column 5.4 describe the keypoint detection for the input image given and the output of the posture detection done by CNN and RCNN respectively:

**Table 5.3 Result of CNN**

| Position      | Keypoint Detected<br>Output Image                                                   | Output Of CNN                                                                                                                                                                                            |
|---------------|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hunch<br>back |   | <pre> processing time1 is 58.31735 secs processing time2 is 153.17488 secs  Person is in Hunchback position! Incorrect Spine Posture </pre>                                                              |
| Straight      |  | <pre> To enable them in other operations, rebuild TensorFlow er flags. processing time1 is 69.10063 secs processing time2 is 151.47656 secs  Person is in Stright position! Correct Spine Posture </pre> |
| Reclined      |  | <pre> processing time1 is 52.14701 secs processing time2 is 94.57633 secs  Person is in Reclined position! Incorrect Spine Posture </pre>                                                                |

The CNN code is tested with a set of images in a different position in different environments. The time taken by the CNN network on an average for calculating each position like a hunchback, reclined or straight is around a minute displaying output with a text alert notification for the position and change of position. As we used ReLU along with CNN, the system reacts more reliable with the convergence of the system is six times more than other methods like sigma and tanh.

**Table 5.4 Result of RCNN**

| Position      | Keypoint detected<br>output image                                                   | Output of RCNN                                                                                                                                                                                                                                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hunch<br>back |   | <pre>start processing... C:\Users\91988\anaconda3\lib\site-packages\torch\func the indexing argument. (Triggered internally at ..\a return_VF.meshgrid(tensors, **kwargs) # type: ign processing time2 is 9.56103</pre> <p>Angle deviated from correct posture: 0.880884973128504 rad<br/>Degree is: 50<br/>Person is in Hunchback position! Incorrect Spine Posture</p> |
| Straight      |  | <pre>C:\Users\91988\anaconda3\lib\site-packages\torch\func pass the indexing argument. (Triggered internally at return_VF.meshgrid(tensors, **kwargs) # type: igno processing time2 is 8.41329</pre> <p>Angle deviated from correct posture: 1.363097644345184 rad<br/>Degree is: 78<br/>Person is in Straight position! Correct Spine Posture</p>                       |
| Reclined      |  | <pre>start processing... C:\Users\91988\anaconda3\lib\site-package the indexing argument. (Triggered interna return_VF.meshgrid(tensors, **kwargs) processing time2 is 8.64860</pre> <p>Angle deviated from correct posture: 1.9868591661721764 rad<br/>Degree is: 114<br/>Person is in Reclined position! Incorrect Spine Posture</p>                                   |

RCNN code is tested and trained with a set of images in different positions in different environments. Time is taken by the RCNN network on an average for calculating each position like a hunchback, reclined or straight is less than a quarter of a minute with displaying text alert notification for the position and change of position. On analysis, it is clear that the RCNN is faster and more accurate when compared with the CNN network as RCNN targets image segmentation for easy classification for lowering the time consumption and program execution.

### 5.3 DISCUSSION

In this proposal, we took CNN and RCNN as an examination tool for comparative study for posture recognition of the computer worker while using their computer. Based on the examination, it was clear that the time taken by the RCNN is very low when compared to CNN. Based on the study, it is proven that compared to other neural networks, Keypoint RCNN is faster as this type of neural network is extracted from faster RCNN and mask RCNN. If more neural networks are taken for study, then we can summarize the preferable neural network tool.

Many existing methodologies for position detection and estimation using a variety of sensors and machine learning has been developed. As a part of research in position monitoring, this research makes a way for the researchers to know the merits of RCNN network over CNN network and how RCNN determines the posture of a person. As research concentrated on contactless and end-user prefers contactless tool for monitoring the posture of the user, Neural network has been studied in a wider range for implementation of the analytical tool. The neural network is preferred more as this method learns to work as Nerves in the human brain so the accuracy in output and quick learning is obtained at a higher rate. Cons of using sensors for measuring the sitting posture are that person is made to sit over sensor placed cushion which may produce discomfort for the persons, there may be lack of accuracy due to external disturbances like cloth material type, cushion position, not sitting properly over the sensors. To avoid this type of disturbances and artifacts in signal inputs, machine learning and artificial intelligence have been used. CNN and RCNN are chosen as they are part of a fundamental neural network that uses mathematical computation for analysis and image processing. Since neural network layers and weight can be added or removed, this system is versatile to user needs. Tough many research works used IoT and other communication techniques for data

transfer, the primary operations are carried out with neural networks.

As this neural network takes the Ear and Hip as key points for detection, this research paper works on the lateral view inputs. The input data are given in the lateral view for proper result outcomes. Further works for this paper can be concentrated on providing position detection even for front view and for real-time code, and an alarm support system can be done for providing more safety.

## CHAPTER 6

### SUMMARY AND CONCLUSION

#### 6.1 CONCLUSION

The spinal posture of the person while working with a computer has been analysed using the CNN and RCNN networks. This research paper is taken as the lifestyle of most of the profession has turned into work from home and the time of interaction with the computer in sitting position has been increased with improper ergonomic workplace set up in a home. To reduce the complication resulting from improper sitting posture and spine, neck related disorders, an analysis tool using neural network have been done. Also, this paper targets to compare two neural networks to segregate the advantages of RCNN and CNN.

A comparative study has been plotted between the time taken by both networks for detecting the posture. Based on the study for multiple images, it is clear that the RCNN has minimal latency for detecting the person's position when compared to CNN. This software supports detecting key points of many people but posture analysis can be done only for one person at a time. This can be extended by iterating the detection part of the code for overall key points that are detected on each individual in the input. The research can further be extended by taking another type of Neural network and comparing their accuracy and latency in detecting the posture.

The aim of this research analysis has been obtained by making a comparative study for both input image, video and real-time live video in two different neural networks namely RCNN and CNN. This is carried out using python language with Keras, TensorFlow, and Torchvision APIs. Google open source along with the MS-COCO training dataset has made this research work easier and less time consuming by making the neural networks trained with these existing materials. Various environmental dataset of lateral position is given as test inputs for both neural networks so that the system performance can be sorted. CNN is performed using the ReLU function which convergence the output at a higher rate. RCNN which is used is a modified version of Mark RCNN and Faster CNN that has a mask for faster elimination of noise and segmentation of image and classification which is performed at a high rate.

## 6.2 FUTURE WORK

Future works for this analysis tool can be done by using other neural networks like ANN, kNN, etc along with RCNN to compare the latency of the network system. Also, the tool can be targeted at extracting posture estimation for multiple people in a single image. This can be obtained by using the iterative method in RCNN and another neural network. For real-time CNN and RCNN posture calculation, implementing an alert text or alarm sound can be done which produces an alert notification for the person when they maintain the wrong posture for a certain amount of time. The threshold for time and angle of deviation between keypoints can be set so that person is monitored properly without any risk or lag of performance of the system.

## REFERENCES

1. Ahmad, J. Sidén, J. and Andersson, H. A proposal of implementation of sitting posture monitoring system for wheelchair utilizing machine learning methods. MDPI. (2021). 21(19): 6349.
2. Arif Reza Anwary, Deniz Cetinkaya, Michael Vassallo and Hamid Bouchachia. Smart-cover: a real time sitting posture monitoring system. Elsevier, Sensors and Actuators. (2021). 317
3. Enrique Piñero-Fuentes, Salvador Canas-Moreno, Antonio Rios-Navarro, Manuel Domínguez-Morale, José Luis Sevillano and Alejandro Linares-Barranco. A Deep-Learning Based Posture Detection System for Preventing Telework-Related Musculoskeletal Disorders. MDPI Applied sciences. (2021). 21(15).
4. Francisco Luna-Perejón, Juan Manuel Montes-Sánchez Lourdes Durán-López, Alberto Vazquez-Baeza, Isabel Beasley-Bohórquez and José L. Sevillano-Ramos. IoT device for sitting posture classification using artificial neural network. MDPI. (2021). 10(15).
5. Gian Domenico Licciardo, Alessandro Russo, Alessandro Naddeo, Nicola Cappetti, Luigi Di Benedetto, Alfredo Rubino and Rosalba Liguori. A resource constrained neural network for the design of embedded human posture recognition systems. MDPI. (2021). 11(11).
6. Hongyi Zhang, Shan Shui, Yuwen Wu and Qiang Yang, Yijun Cai. Posture recognition based on the improved optical diffractive neural network. International Conference on Electronic Information and Communication Technology. pp. (2020). 13-15.
7. Jahnvi Gupta, Nitin Gupta, Mukesh Kumar, and Ritwik Duggal. An introduction to IoT based posture detection and underlying sensor technology. ResearchGate. (2020).
8. Qilong Wan, Haiming Zhao, Jie Li and Peng Xu. Hip positioning and sitting posture recognition based on human sitting pressure image. MDPI. (2021). 21(2).
9. Qisong Hu, Xiaochen Tang and Wei Tang. A smart chair sitting posture recognition system using flex sensors and FPGA implemented artificial neural

network. IEEE sensors journal. (2020). 20(14).

10. Rabia Hasib, Kaleem Nawaz Khan and Miao Yu. Vision-based human posture classification and fall detection using convolutional neural network. International conference on Artificial Intelligence (ICAI). (2021).

11. Rinki Gupta, Devesh Saini and Shubham Mishra. Posture detection using deep learning for time series data. (2020). IEEE.

12. Shumei Zhang and Victor Callaghan. Real-time human posture recognition using an adaptive hybrid classifier - International Journal of Machine Learning and Cybernetics. (2020). 12. pp. 489–499.

13. Sidrah Liaqat, Kia Dashtipour, Kamran Arshad, Khaled Assaleh, Naeem and Ramzan. A hybrid posture detection framework: Integrating machine learning and deep neural networks. IEEE SENSORS JOURNAL. (2021). 21(7).

14. Silvia Caggiari, Peter R. Worsley, Sarah L. Fryer, Joseph Mace and Dan L. Bader. Detection of posture and mobility in individuals at risk of developing pressure ulcers. Elsevier Medical Engineering and Physics. (2021). 91, pp: 39–47.

15. Tatiana Klishkovskaia, Andrey Aksenov Aleksandr Sinitca, Anna Zamansky, Oleg A. Markelov and Dmitry Kaplun. Development of classification algorithms for the detection of postures using non-marker-based motion capture systems. MDPI. (2020). 10 (11).

16. Thanh-Hai Tran, Duc Trung Nguyen and Thanh Phuong Nguyen. Human posture classification from multiple viewpoints and application for fall detection. IEEE. (2020).

17. Wafaa M. Salih Abedi, Dr. Ibraheem Nadher and Dr. Ahmed T. Sadiq. Modification of deep learning technique for face expressions and body postures recognitions. International Journal of Advanced Science and Technology. (2020). 29 (3). pp. 313-320.

18. Wafaa M. Salih Abedi, Dr. Ibraheem Nadher and Dr. Ahmed T. Sadiq. Modified deep learning method for body postures recognition. International Journal of Advanced Science and Technology. (2020). 29(2).

19. Weili Ding, Bo Hu<sup>1</sup>, Han Liu, Xinming Wang and Xiangsheng Huang. Human posture recognition based on multiple features and rule learning. International

Journal of Machine Learning and Cybernetics. (2020). 11(11), pp: 2529–2540.

20. Wenyu Cai, Dongyang Zhao, Meiyang Zhang, Yinan Xu and Zhu Li. Improved self-organizing map-based unsupervised learning algorithm for sitting posture recognition system. MDPI. (2021). 21(18), pp: 6246.

21. Xu Ran, Cong Wang, Yao Xiao, Xuliang Gao, Zhiyuan Zhu and Bin Chen. A portable sitting posture monitoring system based on a pressure sensor array and machine learning. Elsevier. (2021). 331.

22. Yuanhang Feng, Lei Zhang, Yuncheng Zhang, Xiang Li and Quanyin Zhu. An analysis system of students' attendance state in classroom based on human posture recognition. 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science. (2020). 1, pp. 214-217.

## APPENDICES

### CODE FOR CNN

```
import cv2
import math
import time
import numpy as np
import util
from config_reader import config_reader
from scipy.ndimage.filters import gaussian_filter
from model import get_testing_model
tic=0
# visualize
colors = [[255, 0, 0], [255, 85, 0], [255, 170, 0], [255, 255, 0], [170, 255, 0], [85, 255, 0],
          [0, 255, 0], \
          [0, 255, 85], [0, 255, 170], [0, 255, 255], [0, 170, 255], [0, 85, 255], [0, 0, 255],
          [85, 0, 255], \
          [170, 0, 255], [255, 0, 255], [255, 0, 170], [255, 0, 85]]
def process (input_image, params, model_params):
    ''' Start of finding the Key points of full body using Open Pose.'''
    orimg = cv2.imread(input_image) # B,G,R order
    multiplier = [x * model_params['boxsize'] / orimg.shape[0] for x in params['scale_search']]
    heatmap_avg = np.zeros((orimg.shape[0], orimg.shape[1], 19))
    paf_avg = np.zeros((orimg.shape[0], orimg.shape[1], 38))
    for m in range(1):
        scale = multiplier[m]
        imageToTest = cv2.resize(orimg, (0, 0), fx=scale, fy=scale,
interpolation=cv2.INTER_CUBIC)
        imageToTest_padded, pad = util.padRightDownCorner(imageToTest,
model_params['stride'],
                    model_params['padValue'])
        input_img = np.transpose(np.float32(imageToTest_padded[:, :, :, np.newaxis]), (3,0,1,2)) #
required shape (1, width, height, channels)
        output_blobs = model.predict(input_img)
        heatmap = np.squeeze(output_blobs[1]) # output 1 is heatmaps
        heatmap = cv2.resize(heatmap, (0, 0), fx=model_params['stride'], fy=model_params['stride'],
interpolation=cv2.INTER_CUBIC)
        heatmap = heatmap[:imageToTest_padded.shape[0] - pad[2],
:imageToTest_padded.shape[1] - pad[3],
:]
        heatmap = cv2.resize(heatmap, (orimg.shape[1], orimg.shape[0]),
interpolation=cv2.INTER_CUBIC)
        paf = np.squeeze(output_blobs[0]) # output 0 is PAFs
        paf = cv2.resize(paf, (0, 0), fx=model_params['stride'], fy=model_params['stride'],
interpolation=cv2.INTER_CUBIC)
        paf = paf[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape[1] - pad[3],
```

```

:]

    paf = cv2.resize(paf, (orimg.shape[1], orimg.shape[0]), interpolation=cv2.INTER_CUBIC)
    heatmap_avg = heatmap_avg + heatmap / len(multiplier)
    paf_avg = paf_avg + paf / len(multiplier)

all_peaks = [] #To store all the key points which are detected.
peak_counter = 0
prinfTick(1) #prints time required till now.
for part in range(18):
    map_ori = heatmap_avg[:, :, part]
    map = gaussian_filter(map_ori, sigma=3)
    map_left = np.zeros(map.shape)
    map_left[1:, :] = map[:-1, :]
    map_right = np.zeros(map.shape)
    map_right[:-1, :] = map[1:, :]
    map_up = np.zeros(map.shape)
    map_up[:, 1:] = map[:, :-1]
    map_down = np.zeros(map.shape)
    map_down[:, :-1] = map[:, 1:]
    peaks_binary = np.logical_and.reduce(
        (map >= map_left, map >= map_right, map >= map_up, map >= map_down, map
> params['thre1']))
    peaks = list(zip(np.nonzero(peaks_binary)[1], np.nonzero(peaks_binary)[0])) # note reverse
    peaks_with_score = [x + (map_ori[x[1], x[0]],) for x in peaks]
    id = range(peak_counter, peak_counter + len(peaks))
    peaks_with_score_and_id = [peaks_with_score[i] + (id[i],) for i in range(len(id))]

    all_peaks.append(peaks_with_score_and_id)
    peak_counter += len(peaks)

connection_all = []
special_k = []
mid_num = 10
prinfTick(2) #prints time required till now.
print()
position = checkPosition(all_peaks) #check position of spine.
canvas1 = draw(input_image, all_peaks) #show the image.
return canvas1 , position

def draw(input_image, all_peaks):
    canvas = cv2.imread(input_image) # B,G,R order
    for i in range(18):
        for j in range(len(all_peaks[i])):
            cv2.circle(canvas, all_peaks[i][j][0:2], 4, colors[i], thickness=-1)
    return canvas

def checkPosition(all_peaks):
    try:

```

```

f = 0
if (all_peaks[16]):
    a = all_peaks[16][0][0:2] #Right Ear
    f = 1
else:
    a = all_peaks[17][0][0:2] #Left Ear
b = all_peaks[11][0][0:2] # Hip
angle = calcAngle(a,b)
degrees = round(math.degrees(angle))
if (f):
    degrees = 180 - degrees
if (degrees < 80):
    return 1
elif (degrees > 110):
    return -1
else:
    return 0
except Exception as e:
    print("person not in lateral view and unable to detect ears or hip")
#calculate angle between two points with respect to x-axis (horizontal axis)
def calcAngle(a, b):
    try:
        ax, ay = a
        bx, by = b
        if (ax == bx):
            return 1.570796
        return math.atan2(by-ay, bx-ax)
    except Exception as e:
        print("unable to calculate angle")
def calcDistance(a,b): #calculate distance between two points.
    try:
        x1, y1 = a
        x2, y2 = b
        return math.hypot(x2 - x1, y2 - y1)
    except Exception as e:
        print("unable to calculate distance")
def showimage(img): #sometimes opencv will oversize the image when using using `cv2.imshow()`. This
function solves that issue.
    screen_res = 1280, 720 #my screen resolution.
    scale_width = screen_res[0] / img.shape[1]
    scale_height = screen_res[1] / img.shape[0]
    scale = min(scale_width, scale_height)
    window_width = int(img.shape[1] * scale)
    window_height = int(img.shape[0] * scale)
    cv2.namedWindow('image', cv2.WINDOW_NORMAL)

```

```

cv2.resizeWindow('image', window_width, window_height)
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
def printTick(i): #Time calculation to keep a trackm of progress
    toc = time.time()
    print ('processing time%d is %.5f secs' % (i,toc - tic))
if __name__ == '__main__': #main function of the program
    tic = time.time()
    print('start processing...')

    model = get_testing_model()
    model.load_weights('./model/keras/model.h5')

    vi=False
    if(vi == False):
        time.sleep(2)
        params, model_params = config_reader()
        canvas, position= process('./sample_images/(1041).jpg', params, model_params)
        showimage(canvas)
        if (position == 1):
            print("Person is in Hunchback position! Incorrect Spine Posture \n")
        elif (position == -1):
            print ("Person is in Reclined position! Incorrect Spine Posture \n")
        else:
            print("Person is in Stright position! Correct Spine Posture \n")

```

## CODE FOR RCNN

```

import cv2
import math
import time
import numpy as np
import util
from config_reader import config_reader
from scipy.ndimage.filters import gaussian_filter
from model import get_testing_model
from torchvision import transforms as T
import torchvision
import matplotlib.pyplot as plt
tic=0
# visualize
colors = [[255, 0, 0], [255, 85, 0], [255, 170, 0], [255, 255, 0], [170, 255, 0], [85, 255, 0],
          [0, 255, 0], \
          [0, 255, 85], [0, 255, 170], [0, 255, 255], [0, 170, 255], [0, 85, 255], [0, 0, 255],

```

```

[85, 0, 255], \
[170, 0, 255], [255, 0, 255], [255, 0, 170], [255, 0, 85]]

def process (input_image, params, model_params):
    orimg = cv2.imread(input_image) # B,G,R order
    img=orimg
    transform = T.Compose([T.ToTensor()])
    img_tensor = transform(orimg)
    output = model([img_tensor])[0]
    printTick(2) #prints time required till now.
    cmap = plt.get_cmap('rainbow')
    # create a copy of the image
    img_copy = orimg.copy()
    all_keypoints=output["keypoints"]
    confs=output["scores"]
    conf_threshold=0.9
    all_scores=output["keypoints_scores"]
    keypoint_threshold=2
    color_id = np.arange(1,255, 255//len(all_keypoints)).tolist()[::-1]
    # iterate for every person detected
    for person_id in range(len(all_keypoints)):
        # check the confidence score of the detected person
        if confs[person_id]>conf_threshold:
            # grab the keypoint-locations for the detected person
            keypoints = all_keypoints[person_id, ...]
            # grab the keypoint-scores for the keypoints
            scores = all_scores[person_id, ...]
            # iterate for every keypoint-score
            for kp in range(len(scores)):
                # check the confidence score of detected keypoint
                if scores[kp]>keypoint_threshold:
                    # convert the keypoint float-array to a python-list of intergers
                    keypoint = tuple(map(int, keypoints[kp, :2].detach().numpy().tolist()))
                    # pick the color at the specific color-id
                    color = tuple(np.asarray(cmap(color_id[person_id])[::-1])*255)
                    # draw a cirle over the keypoint location
                    cv2.circle(img_copy, keypoint, 10, color, -1)
            print("Key Points are: \n",keypoints)
            position = checkPosition(keypoints) #check position of spine.
            #print("Position:", position)
            return img_copy , position
def draw(input_image, all_peaks):
    canvas = cv2.imread(input_image) # B,G,R order
    for i in range(18):
        for j in range(len(all_peaks[i])):
            cv2.circle(canvas, all_peaks[i][j][0:2], 4, colors[i], thickness=-1)

```

```

    return canvas
def checkPosition(all_peaks):
    f = 0
    if not (all_peaks[4] is None):
        a = all_peaks[4][0:2] #Right Ear
        f = 1
    else:
        a = all_peaks[3][0:2] #Left Ear
    b = all_peaks[11][0:2] # Hip
    angle = calcAngle(a,b)
    print("Angle deviated from correct posture:", angle ,"rad")
    degrees = round(math.degrees(angle))
    print("Degree is:", degrees)
    if (f):
        degrees = 180 - degrees
    if (degrees < 80):
        return -1
    elif (degrees > 110):
        return 1
    else:
        return 0
#calculate angle between two points with respect to x-axis (horizontal axis)
def calcAngle(a, b):
    try:
        ax, ay = a
        bx, by = b
        if (ax == bx):
            return 1.570796
        return math.atan2(by-ay, bx-ax)
    except Exception as e:
        print("unable to calculate angle")
def checkHandFold(all_peaks):
    try:
        if (all_peaks[3][0][0:2]):
            try:
                if (all_peaks[4][0][0:2]):
                    distance = calcDistance(all_peaks[3][0][0:2],all_peaks[4][0][0:2])
#distance between right arm-joint and right palm.
                    armdist = calcDistance(all_peaks[2][0][0:2], all_peaks[3][0][0:2])
#distance between left arm-joint and left palm.
                    if (distance < (armdist + 100) and distance > (armdist - 100) ):
#this value 100 is arbitrary. this shall be replaced with a calculation which can adjust to different sizes of
people.

                    print("Not Folding Hands")
            else:

```

```

        print("Folding Hands")
    except Exception as e:
        print("Folding Hands")
except Exception as e:
    try:
        if(all_peaks[7][0][0:2]):
            distance = calcDistance( all_peaks[6][0][0:2] ,all_peaks[7][0][0:2])
            armdist = calcDistance(all_peaks[6][0][0:2], all_peaks[5][0][0:2])
            # print(distance)
            if (distance < (armdist + 100) and distance > (armdist - 100)):
                print("Not Folding Hands")
            else:
                print("Folding Hands")
        except Exception as e:
            print("Unable to detect arm joints")
def calcDistance(a,b): #calculate distance between two points.
    try:
        x1, y1 = a
        x2, y2 = b
        return math.hypot(x2 - x1, y2 - y1)
    except Exception as e:
        print("unable to calculate distance")
def showimage(img): #sometimes opencv will oversize the image when using using `cv2.imshow()`. This
function solves that issue.
    screen_res = 1280, 720 #my screen resolution.
    scale_width = screen_res[0] / img.shape[1]
    scale_height = screen_res[1] / img.shape[0]
    scale = min(scale_width, scale_height)
    window_width = int(img.shape[1] * scale)
    window_height = int(img.shape[0] * scale)
    cv2.namedWindow('image', cv2.WINDOW_NORMAL)
    cv2.resizeWindow('image', window_width, window_height)
    cv2.imshow('image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
def printTick(i): #Time calculation to keep a trackm of progress
    toc = time.time()
    print ('processing time%d is %.5f' % (i,toc - tic))
if __name__ == '__main__': #main function of the program
    tic = time.time()
    print('start processing...')
    model = torchvision.models.detection.keypointcnn_resnet50_fpn(pretrained=True)
    # call the eval() method to prepare the model for inference mode.
    model.eval()
    vi=False

```

```

if(vi == False):
    time.sleep(2)
    params, model_params = config_reader()
    canvas, position= process('./sample_images/(1006).jpg', params, model_params)
    showimage(canvas)
    if (position == 1):
        print("Person is in Hunchback position! Incorrect Spine Posture \n")
    elif (position == -1):
        print ("Person is in Reclined position! Incorrect Spine Posture \n")
    else:
        print("Person is in Straight position! Correct Spine Posture \n")

```

## CODE FOR REALTIME CNN

```

import cv2
import math
import time
import numpy as np
import util
from config_reader import config_reader
from scipy.ndimage.filters import gaussian_filter
from model import get_testing_model
from torchvision import transforms as T
import torchvision
import matplotlib.pyplot as plt
tic=0
# visualize
colors = [[255, 0, 0], [255, 85, 0], [255, 170, 0], [255, 255, 0], [170, 255, 0], [85, 255, 0],
          [0, 255, 0], \
          [0, 255, 85], [0, 255, 170], [0, 255, 255], [0, 170, 255], [0, 85, 255], [0, 0, 255],
          [85, 0, 255], \
          [170, 0, 255], [255, 0, 255], [255, 0, 170], [255, 0, 85]]
def process (input_image, params, model_params):
    orimg = input_image # B,G,R order
    img=orimg
    transform = T.Compose([T.ToTensor()])
    img_tensor = transform(orimg)
    output = model([img_tensor])[0]
    printTick(2) #prints time required till now.
    cmap = plt.get_cmap('rainbow')
    # create a copy of the image
    img_copy = orimg.copy()
    all_keypoints=output["keypoints"]
    confs=output["scores"]
    conf_threshold=0.9

```

```

all_scores=output["keypoints_scores"]
keypoint_threshold=2
color_id = np.arange(1,255, 255//len(all_keypoints)).tolist()[::-1]
# iterate for every person detected
for person_id in range(len(all_keypoints)):
    # check the confidence score of the detected person
    if confs[person_id]>conf_threshold:
        # grab the keypoint-locations for the detected person
        keypoints = all_keypoints[person_id, ...]
        # grab the keypoint-scores for the keypoints
        scores = all_scores[person_id, ...]
        # iterate for every keypoint-score
        for kp in range(len(scores)):
            # check the confidence score of detected keypoint
            if scores[kp]>keypoint_threshold:
                # convert the keypoint float-array to a python-list of intergers
                keypoint = tuple(map(int, keypoints[kp, :2].detach().numpy().tolist()))
                # pick the color at the specific color-id
                color = tuple(np.asarray(cmap(color_id[person_id])[::-1])*255)
                # draw a circle over the keypoint location
                cv2.circle(img_copy, keypoint, 10, color, -1)
        print(keypoints)
        position = checkPosition(keypoints) #check position of spine.
        print(position)
        return img_copy
    return img_copy
def draw(input_image, all_peaks):
    canvas = cv2.imread(input_image) # B,G,R order
    for i in range(18):
        for j in range(len(all_peaks[i])):
            cv2.circle(canvas, all_peaks[i][j][0:2], 4, colors[i], thickness=-1)
    return canvas
def checkPosition(all_peaks):
    try:
        f = 0
        if not (all_peaks[4] is None):
            a = all_peaks[4][0:2] #Right Ear
            f = 1
        else:
            a = all_peaks[3][0:2] #Left Ear
        b = all_peaks[11][0:2] # Hip
        angle = calcAngle(a,b)
        print(angle)
        degrees = round(math.degrees(angle))
        print(degrees)

```

```

    if (f):
        degrees = 180 - degrees
    if (degrees<70):
        return 1
    elif (degrees > 110):
        return -1
    else:
        return 0
except Exception as e:
    print("person not in lateral view and unable to detect ears or hip")
#calculate angle between two points with respect to x-axis (horizontal axis)
def calcAngle(a, b):
    try:
        ax, ay = a
        bx, by = b
        if (ax == bx):
            return 1.570796
        return math.atan2(by-ay, bx-ax)
    except Exception as e:
        print("unable to calculate angle")
def checkHandFold(all_peaks):
    try:
        if (all_peaks[3][0][0:2]):
            try:
                if (all_peaks[4][0][0:2]):
                    distance = calcDistance(all_peaks[3][0][0:2],all_peaks[4][0][0:2])
#distance between right arm-joint and right palm.
                    armdist = calcDistance(all_peaks[2][0][0:2], all_peaks[3][0][0:2])
#distance between left arm-joint and left palm.
                    if (distance < (armdist + 100) and distance > (armdist - 100) ):
#this value 100 is arbitrary. this shall be replaced with a calculation which can adjust to different sizes of
people.
                        print("Not Folding Hands")
                    else:
                        print("Folding Hands")
            except Exception as e:
                print("Folding Hands")
    except Exception as e:
        try:
            if(all_peaks[7][0][0:2]):
                distance = calcDistance( all_peaks[6][0][0:2] ,all_peaks[7][0][0:2])
                armdist = calcDistance(all_peaks[6][0][0:2], all_peaks[5][0][0:2])
                # print(distance)
                if (distance < (armdist + 100) and distance > (armdist - 100)):
                    print("Not Folding Hands")

```

```

        else:
            print("Folding Hands")
    except Exception as e:
        print("Unable to detect arm joints")
def calcDistance(a,b): #calculate distance between two points.
    try:
        x1, y1 = a
        x2, y2 = b
        return math.hypot(x2 - x1, y2 - y1)
    except Exception as e:
        print("unable to calculate distance")
def showimage(img): #sometimes opencv will oversize the image when using using `cv2.imshow()`. This
function solves that issue.
    screen_res = 1280, 720 #my screen resolution.
    scale_width = screen_res[0] / img.shape[1]
    scale_height = screen_res[1] / img.shape[0]
    scale = min(scale_width, scale_height)
    window_width = int(img.shape[1] * scale)
    window_height = int(img.shape[0] * scale)
    cv2.namedWindow('image', cv2.WINDOW_NORMAL)
    cv2.resizeWindow('image', window_width, window_height)
    cv2.imshow('image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
def printTick(i):
    toc = time.time()
    print ('processing time%d is %.5f' % (i,toc - tic))
if __name__ == '__main__':
    tic = time.time()
    print('start processing...')
    model = torchvision.models.detection.keypointrcnn_resnet50_fpn(pretrained=True)
    # call the eval() method to prepare the model for inference mode.
    model.eval()
    cap=cv2.VideoCapture(0)
    vi=cap.isOpened()
    if(vi == True):
        cap.set(100,160)
        cap.set(200,120)
        time.sleep(2)
        while(1):
            tic = time.time()
            ret,frame=cap.read()
            params, model_params = config_reader()
            canvas = process(frame, params, model_params)
            cv2.imshow("capture",canvas)

```

```

        if cv2.waitKey(1) & 0xFF==ord('q'):
            break
    cap.release()
else:
    print("unable to open camera")
cv2.destroyAllWindows()

```

## CODE FOR REALTIME RCNN

```

import cv2
import math
import time
import numpy as np
import util
from config_reader import config_reader
from scipy.ndimage.filters import gaussian_filter
from model import get_testing_model
tic=0
# visualize
colors = [[255, 0, 0], [255, 85, 0], [255, 170, 0], [255, 255, 0], [170, 255, 0], [85, 255, 0],
          [0, 255, 0], \
          [0, 255, 85], [0, 255, 170], [0, 255, 255], [0, 170, 255], [0, 85, 255], [0, 0, 255],
          [85, 0, 255], \
          [170, 0, 255], [255, 0, 255], [255, 0, 170], [255, 0, 85]]
def process (input_image, params, model_params):
    """ Start of finding the Key points of full body using Open Pose."""
    orimg = input_image # B,G,R order
    multiplier = [x * model_params['boxsize'] / orimg.shape[0] for x in params['scale_search']]
    heatmap_avg = np.zeros((orimg.shape[0], orimg.shape[1], 19))
    paf_avg = np.zeros((orimg.shape[0], orimg.shape[1], 38))
    for m in range(1):
        scale = multiplier[m]
        imageToTest = cv2.resize(orimg, (0, 0), fx=scale, fy=scale, interpolation=cv2.INTER_CUBIC)
        imageToTest_padded, pad = util.padRightDownCorner(imageToTest, model_params['stride'],
                                                            model_params['padValue'])
        input_img = np.transpose(np.float32(imageToTest_padded[:, :, :, np.newaxis]), (3,0,1,2)) # required shape
        (1, width, height, channels)
        output_blobs = model.predict(input_img)
        heatmap = np.squeeze(output_blobs[1]) # output 1 is heatmaps
        heatmap = cv2.resize(heatmap, (0, 0), fx=model_params['stride'], fy=model_params['stride'],
                               interpolation=cv2.INTER_CUBIC)
        heatmap = heatmap[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape[1] - pad[3],
                           :]
        heatmap = cv2.resize(heatmap, (orimg.shape[1], orimg.shape[0]), interpolation=cv2.INTER_CUBIC)
        paf = np.squeeze(output_blobs[0]) # output 0 is PAFs

```

```

    paf = cv2.resize(paf, (0, 0), fx=model_params['stride'], fy=model_params['stride'],
                    interpolation=cv2.INTER_CUBIC)
    paf = paf[:imageToTest_padded.shape[0] - pad[2], :imageToTest_padded.shape[1] - pad[3], :]
    paf = cv2.resize(paf, (orlimg.shape[1], orlimg.shape[0]), interpolation=cv2.INTER_CUBIC)
    heatmap_avg = heatmap_avg + heatmap / len(multiplier)
    paf_avg = paf_avg + paf / len(multiplier)
all_peaks = [] #To store all the key points which are detected.
peak_counter = 0

printTick(1) #prints time required till now.
for part in range(18):
    map_ori = heatmap_avg[:, :, part]
    map = gaussian_filter(map_ori, sigma=3)
    map_left = np.zeros(map.shape)
    map_left[1:, :] = map[:-1, :]
    map_right = np.zeros(map.shape)
    map_right[:-1, :] = map[1:, :]
    map_up = np.zeros(map.shape)
    map_up[:, 1:] = map[:, :-1]
    map_down = np.zeros(map.shape)
    map_down[:, :-1] = map[:, 1:]
    peaks_binary = np.logical_and.reduce(
        (map >= map_left, map >= map_right, map >= map_up, map >= map_down, map > params['thre1']))
    peaks = list(zip(np.nonzero(peaks_binary)[1], np.nonzero(peaks_binary)[0])) # note reverse
    peaks_with_score = [x + (map_ori[x[1], x[0]],) for x in peaks]
    id = range(peak_counter, peak_counter + len(peaks))
    peaks_with_score_and_id = [peaks_with_score[i] + (id[i],) for i in range(len(id))]
    all_peaks.append(peaks_with_score_and_id)
    peak_counter += len(peaks)
connection_all = []
special_k = []
mid_num = 10
printTick(2) #prints time required till now.
canvas = frame# B,G,R order
for i in range(18): #drawing all the detected key points.
    for j in range(len(all_peaks[i])):
        cv2.circle(canvas, all_peaks[i][j][0:2], 4, colors[i], thickness=-1)
print()
position = checkPosition(all_peaks)
checkKneeling(all_peaks)
checkHandFold(all_peaks)
print()
print()
return canvas
def checkPosition(all_peaks):

```

```

try:
    f = 0
    if (all_peaks[16]):
        a = all_peaks[16][0][0:2] #Right Ear
        f = 1
    else:
        a = all_peaks[17][0][0:2] #Left Ear
        b = all_peaks[11][0][0:2] # Hip
        angle = calcAngle(a,b)
        degrees = round(math.degrees(angle))
        if (f):
            degrees = 180 - degrees
        if (degrees<70):
            return -1
        elif (degrees > 110):
            return 1
        else:
            return 0
except Exception as e:
    print("person not in lateral view and unable to detect ears or hip")
def calcAngle(a, b):
    try:
        ax, ay = a
        bx, by = b
        if (ax == bx):
            return 1.570796
        return math.atan2(by-ay, bx-ax)
    except Exception as e:
        print("unable to calculate angle")
def checkHandFold(all_peaks):
    try:
        if (all_peaks[3][0][0:2]):
            try:
                if (all_peaks[4][0][0:2]):
                    distance = calcDistance(all_peaks[3][0][0:2],all_peaks[4][0][0:2]) #distance between right arm-
joint and right palm.
                    armdist = calcDistance(all_peaks[2][0][0:2], all_peaks[3][0][0:2]) #distance between left arm-joint
and left palm.
                    if (distance < (armdist + 100) and distance > (armdist - 100) ): #this value 100 is arbitrary. this
shall be replaced with a calculation which can adjust to different sizes of people.
                        print("Not Folding Hands")
                    else:
                        print("Folding Hands")
            except Exception as e:
                print("Folding Hands")

```

```

except Exception as e:
    try:
        if(all_peaks[7][0][0:2]):
            distance = calcDistance( all_peaks[6][0][0:2] ,all_peaks[7][0][0:2])
            armdist = calcDistance(all_peaks[6][0][0:2], all_peaks[5][0][0:2])
            if (distance < (armdist + 100) and distance > (armdist - 100)):
                print("Not Folding Hands")
            else:
                print("Folding Hands")
    except Exception as e:
        print("Unable to detect arm joints")
def calcDistance(a,b): #calculate distance between two points.
    try:
        x1, y1 = a
        x2, y2 = b
        return math.hypot(x2 - x1, y2 - y1)
    except Exception as e:
        print("unable to calculate distance")
def checkKneeling(all_peaks):
    f = 0
    if (all_peaks[16]):
        f = 1
    try:
        if(all_peaks[10][0][0:2] and all_peaks[13][0][0:2]):
            rightankle = all_peaks[10][0][0:2]
            leftankle = all_peaks[13][0][0:2]
            hip = all_peaks[11][0][0:2]
            leftangle = calcAngle(hip,leftankle)
            leftdegrees = round(math.degrees(leftangle))
            rightangle = calcAngle(hip,rightankle)
            rightdegrees = round(math.degrees(rightangle))
            if (f == 0):
                leftdegrees = 180 - leftdegrees
                rightdegrees = 180 - rightdegrees
            if (leftdegrees > 60 and rightdegrees > 60): # 60 degrees is trail and error value here. We can tweak this
                accordingly and results will vary.
                print ("Both Legs are in Kneeling")
            elif (rightdegrees > 60):
                print ("Right leg is kneeling")
            elif (leftdegrees > 60):
                print ("Left leg is kneeling")
            else:
                print ("Not kneeling")
    except IndexError as e:
        try:

```

```

    if (f):
        a = all_peaks[10][0][0:2] # if only one leg (right leg) is detected
    else:
        a = all_peaks[13][0][0:2] # if only one leg (left leg) is detected
    b = all_peaks[11][0][0:2] #location of hip
    angle = calcAngle(b,a)
    degrees = round(math.degrees(angle))
    if (f == 0):
        degrees = 180 - degrees
    if (degrees > 60):
        print ("Both Legs Kneeling")
    else:
        print("Not Kneeling")
except Exception as e:
    print("legs not detected")

def printTick(i):
    toc = time.time()
    print ('processing time%d is %.5f' % (i,toc - tic))
if __name__ == '__main__':
    tic = time.time()
    print('start processing...')
    model = get_testing_model()
    model.load_weights('./model/keras/model.h5')
    cap=cv2.VideoCapture(0)
    vi=cap.isOpened()
    if(vi == True):
        cap.set(100,160)
        cap.set(200,120)
        time.sleep(2)
    while(1):
        tic = time.time()
        ret,frame=cap.read()
        params, model_params = config_reader()
        canvas = process(frame, params, model_params)
        cv2.imshow("capture",canvas)
        if cv2.waitKey(1) & 0xFF==ord('q'):
            break
        cap.release()
    else:
        print("unable to open camera")
cv2.destroyAllWindows()

```