Human Pose Estimation using Deep Learning

A Project Report

Submitted to Jorhat Engineering College

in partial fulfillment of requirements for the award of degree

Bachelor of Technology

in

Computer Science and Engineering

by

Akangkshya Pathak(180712707001)

Apeksha Modi(180710007005)

Masoom Konwar(180710007034)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JORHAT ENGINEERING COLLEGE JORHAT

ASSAM

August 2021

CERTIFICATE FROM INSTITUTE

This is to certify that the report entitled **Human Pose Estimation using Deep Learning** submitted by **Akangkshya Pathak** (180712707001), **Apeksha Modi** (180710007005) and **Masoom Konwar** (180710007005) under the guidance of Mr. Biswajit Sarmah in partial fulfillment of the requirements for the Degree of Computer Science And Engineering Final Year Project is a bona fide work carried out by them.

Signature of head of department:

Dr. Rupam Baruah
Professor and Head
Dept.of CSE
Jorhat Engineering College
Jorhat

CERTIFICATE FROM PROJECT GUIDE

This is to certify that the report entitled **Human Pose Estimation using Deep Learning** submitted by **Akangkshya Pathak** (180712707001), **Apeksha Modi** (180710007005) and **Masoom Konwar** (180710007005) under the guidance of Mr. Biswajit Sarmah in partial fulfillment of the requirements for the Degree of Computer Science And Engineering Final Year Project is a bona fide work carried out by them.

Signature of Guide:

Mr. Biswajit Sarmah
Asst. Professor
Dept.of CSE
Jorhat Engineering College
Jorhat

DECLARATION

We here by declared that the project work entitled **Human Pose Estimation using Deep Learning** ,submitted to the Jorhat Engineering College is an original work done with the guidance of Mr. Biswajit Sarmah, Assistant Professor, Dept. of Computer Science and Engineering, Jorhat Engineering College, and is submitted in partial fulfillment of the requirements for the Degree of B.E in Computer Science and Engineering. The result embodied in the report has not been submitted to any other University or Institute for the award of any degree or diploma.

Akangkshya Pathak

Apeksha Modi

Masoom Kownwar

Jorhat

March 2022

Acknowledgement

We would like to express our heartfull gratitude to all the people who have help us from time to time during the course of this project without whom this work would never been accomplished. First and foremost we are very grateful to our HOD, **Dr. Rupam Baruah** Sir for including **Human Pose Estimation using Deep Learning** project in our course which greatly helps the students to enhance their practical development and coding skills and project guide **Mr. Biswajit Sarmah** Sir for his excellent idea and for having accepted our project. We are deeply indebted to both of them for their support, vision, advice, encouragement, passion and highly positive attitude for the project without whom our project would not been through. At last we shall remain indebted to our classmate and staffs of the Computer Science and Engineering Department for rendering their cooperation and everybody who have directly helped us in carrying out the project.

Akangkshya Pathak
Apeksha Modi
Masoom Konwar

Abstract

Human Pose Estimation is a way of identifying and classifying the joints in the human body. It is a way to capture a set of coordinates for each joint for example head,neck, shoulder , hips, knees, ankles,hands etc which is known as a key point that can describe a pose of a person. The connection between these points is known as a pair and a skeleton is created by connecting these points in a hierarchical manner. In this project, we are trying to use Deep Learning, a method proposed by Alexander Toshev and Christian Szegedy published in the paper named “DeepPose: Human Pose Estimation via Deep Neural Networks” which uses DNN based regression to give coordinates of joints as output by taking images of humans doing some pose as an output. We tried to implement their model along with that we tried to make some of our own adjustment between the layers of the neural network to experiment with the outcome and accuracy of our goal is to improve the accuracy of Human pose estimation models and compare different CNN architectures with each other to see which one performs better in this model. While implementing the model, we faced issues with the pre existing datasets like LSP, FLIC for human pose estimation, hence we also designed a new data annotation tool which will help to prepare dataset specifically designed for this problem which may result improvement upon the overall accuracy of the various techniques. After which we implemented other CNN architectures, namely ResNet 34 and AlexNet to check the accuracy against the DeepPose CNN architecture and compared each model to find the best one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Pose Estimation problem

The problem of human pose estimation, defined as the problem of localization of human joints, has enjoyed substantial attention in the computer vision community. Some of the challenges of this problem – strong articulations, small and barely visible joints, occlusions and the need to capture the context. This is a very new topic in the field of computer vision although some of the papers previously have tried to solve the problem but the research is still ongoing and a lot of work is still yet to be done. Some of the methods that are used to solve this problem are part based models where they try to localize each part and then estimate the joints in the intersecting parts and DNN based methods like deep-pose and some other techniques that use heat-map of the joint area and optical flow to locate the joint.

1.2 Applications of Human Pose Estimation

Solving the human pose estimation problem will lead to use of it in various fields like motion capture and animation where users will be able to capture ”motion capture data” from their smartphone cameras and transfer that data to computer generated characters and also it will be applicable to create full body AR filters. It can also be used to control robotic or cybernetic arms and complete humanoid bodies using video feed of human as a controller.

1.3 Dataset

After researching various sources and reading several papers on this topic we found out that there are primarily two datasets that are used in almost every model to benchmark the performance of the models. They are FLIC and LSP, both of which are available on public domain so we downloaded and tried to understand the contents of the data.

1.3.1 FLIC

FLIC is an abbreviation for Frames Labeled In Cinema, This dataset contains 5003 stills or screenshots from various hollywood movies. These images contain single and multiple person and a mat file that comes with it contains coordinates of every visible joins of the every person present in the scene.

1.3.2 LSP

This dataset contains 2000 pose annotated images of mostly sports people. The images have been scaled such that the most prominent person is roughly 150 pixels in length. Each image has been annotated with 14 joint locations. Left and right joints are consistently labelled from a person-centric viewpoint.

1.4 Data Annotation Tool

In machine learning, data annotation is the process of labelling data to show the outcome you want your machine learning model to predict. We are marking - labelling, tagging, transcribing, or processing - a dataset with the features we want our machine learning system to learn to recognize. Once our model is deployed, we want it to recognize those features on its own and make a decision or take some action as a result. Annotated data reveals features that will train our algorithms to identify the same features in data that has not been annotated. Data annotation is used in supervised learning and hybrid, or semi-supervised, machine learning models that involve supervised learning.

1.5 ResNet

ResNet, short for Residual Network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper “Deep Residual Learning for Image Recognition”. Mostly in order to solve a complex problem, we stack some additional layers in the Deep Neural Networks which results in improved accuracy and performance. The intuition behind adding more layers is that these layers progressively learn more complex features. For example, in case of recognising images, the first layer may learn to detect edges, the second layer may learn to identify textures and similarly the third layer can learn to detect objects and so on. But it has been found that there is a maximum threshold for depth with the traditional Convolutional neural network model Residual networks have become quite popular for image recognition and classification tasks because of their ability to solve vanishing and exploding gradients when adding more layers to an already deep neural network.

1.6 AlexNet

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky’s Ph.D. advisor. In AlexNet’s first layer, the convolution window shape is 11×11. Since most images in ImageNet are more than ten times higher and wider than the MNIST images, objects in ImageNet data tend to occupy more pixels. Consequently, a larger convolution window is needed to capture the object. The convolution window shape in the second layer is reduced to 5×5, followed by 3×3. In addition, after the first, second, and fifth convolutional layers, the network adds maximum pooling layers with a window shape of 3×3 and a stride of 2. Moreover, AlexNet has ten times more convolution channels than LeNet. After the last convolutional layer there are two fully-connected layers with 4096 outputs. These two huge fully-connected layers produce model parameters of nearly 1 GB. AlexNet uses the ReLU activation function.

Chapter 2

Literature Review

2.1 Flowing ConvNets for Human Pose Estimation in Videos - 2015

Authors : Tomas Pfister, James Charles and Andrew Zisserman

In the technique proposed in this paper, a Convolution Network architecture is proposed that is able to benefit from temporal context by combining information across the multiple frames using optical flow. To this end a new network architecture is proposed which regresses a confidence heatmap of joint position predictions; incorporates optical flow at a mid-layer to align heatmap predictions from neighbouring frames; and includes a final parametric pooling layer which learns to combine the aligned heatmaps into a pooled confidence map.

2.1.1 Advantages

1. This architecture outperforms neural networks that regress joint coordinates directly.
2. This paper improves the accuracy from previous methods.

2.1.2 Disadvantages

1. The algorithm implemented in this paper cannot identify z coordinates in photos.

2.2 ImageNet Classification with Deep Convolutional Neural Networks

Authors: Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton

In the technique proposed in this paper, we take an image as an input and use a 8 layer deep neural network to identify the location of joints in the body of the person depicted. Then we take each of these coordinates and draw a bounding box around each of them. After this we take each of these bounding boxes and apply another deep neural network in them to find the coordinate of these joints. The coordinate found is taken as y predicted and we train it with the actual value of y as we have got from the dataset using a neural network in order to improve its efficiency.

2.3 DeepPose: Human Pose Estimation via Deep Neural Networks - 2014

Authors : Alexander Toshev and Christian Szegedy

In the technique proposed in this paper, we take an image as an input and use a 7 layer deep neural network to identify the location of joints in the body of the person depicted. Then we take each of these coordinates and draw a bounding box around each of them. After this we take each of these bounding boxes and apply another deep neural network in them to find the coordinate of these joints. The coordinate found is taken as y predicted and we train it with the actual value of y as we have got from the dataset using a neural network in order to improve its efficiency

2.3.1 Advantages

1. The algorithm mentioned in the paper is easy to understand.
2. The algorithm mentioned in the paper is easy to implement.
3. This technique follows a simple approach.

2.3.2 Disadvantages

1. The algorithm implemented in this paper cannot identify z coordinates in photos.
2. The accuracy of the model is low compared to other techniques.

Chapter 3

Methodology

The steps followed to create the model are:

3.1 Implementation of DeepPose Model

3.1.1 Importing the libraries

For the implementation of this paper, we use the following libraries :

TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

MatplotLib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

OpenCv

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

OS

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

Scipy

SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely.

Sickit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

Pickle

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.)

3.1.2 Reading the image

We read the image using “imread” from openCV. We resize the image to a suitable size to optimize memory usage and orient the image to a correct way if it isn’t oriented properly.

3.1.3 Preprocessing

As preprocessing step we first resized the image to 100x100 pixels to provide a reasonable amount of parameters to the neural network. Then we found that after resizing the images the coordinates denoting the joints are no longer in the place where they should be. Hence, we used normalization so that the joints are corrected according to the new dimensions of the images and after that the data is ready to be fed into the CNN model.

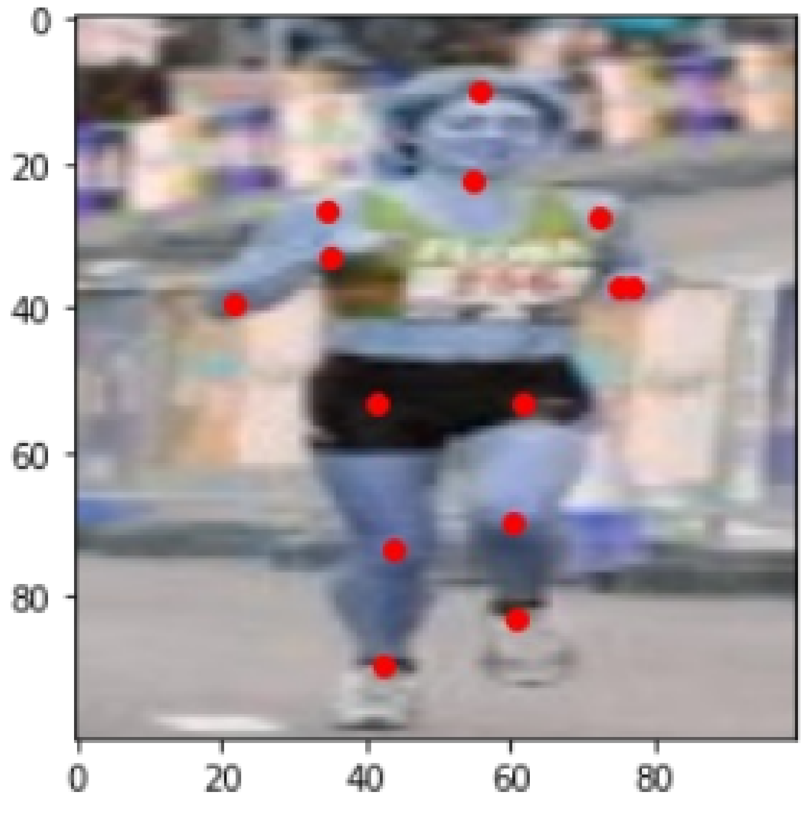


Fig 1

3.1.4 Normalisation of coordinates

We used lsp dataset to train the model. The dataset contained 2000 labeled images with a dictionary containing a matrix of 2000x14x3 dimension, where each image has 14 coordinates of the joints in that matrix but every image has their own size and shape and coordinates are also labeled according their shape. As we could not fit images with different shape to the CNN model, we resized and reshaped the image to 100x100 pixels and then used it as an input layer but because joint coordinates are labeled according to the shapes we had to normalize the coordinates of the joints to fit the new image shape so that the model does not predict wrong coordinates. We used the following formula:

$$coordinateX = \frac{coordinate_X}{dimension_X} * imagesize$$

$$coordinateY = \frac{coordinate_Y}{dimension_Y} * imagesize$$

Here,

Coordinate$_X$ $= XCoordinateofthejoint$

$Coordinate_Y = Ycoordinateofthejoint$

$Dimension_X = Widthoftheimage$

$Dimension_Y = Heightoftheimage$

$Imagesize = Sizeoftheinputimage$

3.1.5 Model Description

We implemented 14 separate CNN models to identify the different joints of the human body. The model contains 7 layers, denoted by C, a convoltional layer, Bn, a batch normalization layer, A, a relu activation layer, P, a max pooling layer and F, a fully connected layer. The layers C and F are the only layers containing learnable parameters, while the rest of the layers are parameter free.

For C layers, the size is defined as:

size = width x height x depth

where,

the first two dimensions have a spatial meaning. The dimension named depth defines the number of filters. If we write the size of each layer in parenthesis, then the network can be described concisely as:

C (23 x 23 x 48) - Bn - A - P - C (11 x 11 x 128) - Bn - A - P - C (5 X 5 X 192) - A - C (5 X 5 X 128) - A - P - F (4096) - F (4096)

The filter size for the first two C layers is 11 x 11 and for the remaining two layers, the filter size is 3 x 3. Same padding is used to preserve the size of all convolutional layers. Max pooling is applied after three layers and contributes to increased performance despite the reduction is resolution. The imput to the net is an image of 100 x 100 x 3 dimensions which, via a stride of 4 is fed into the network. The output layer consists of two neurons. In this output layer, relu is used as an activation function and output from this layer is considered as final x and y coordinated of the model.

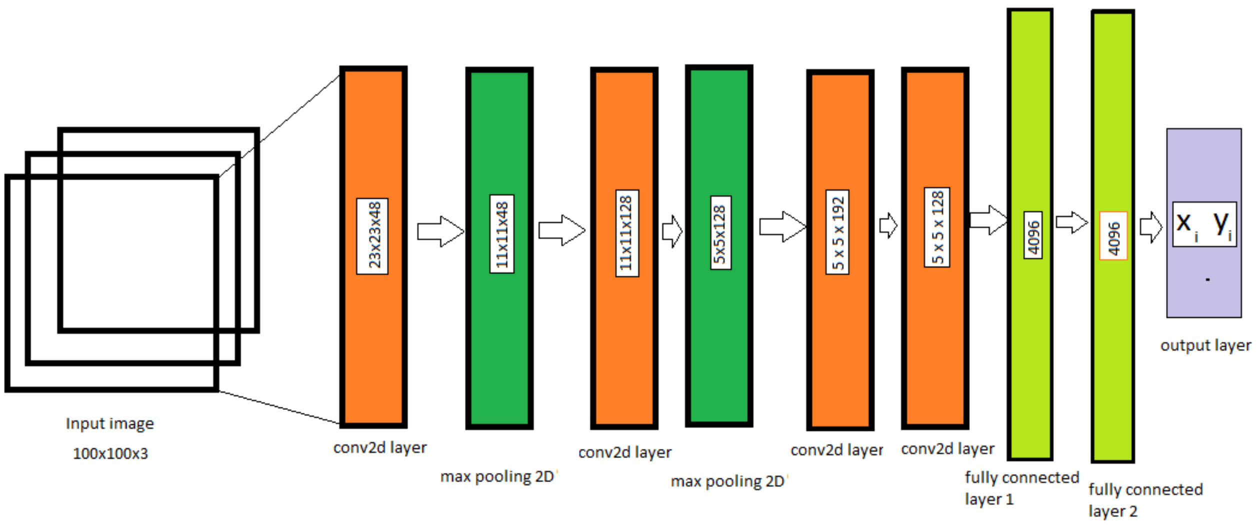
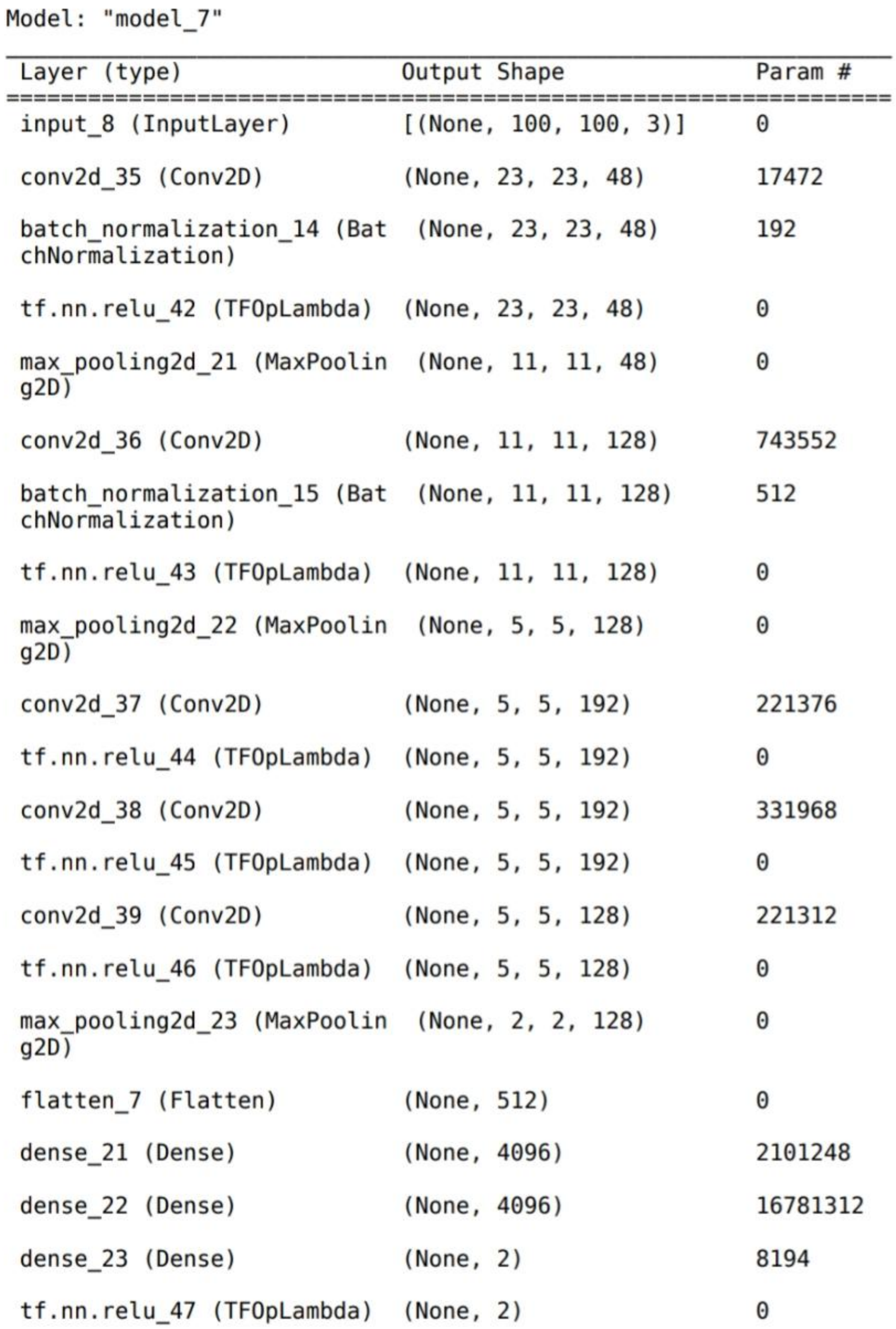
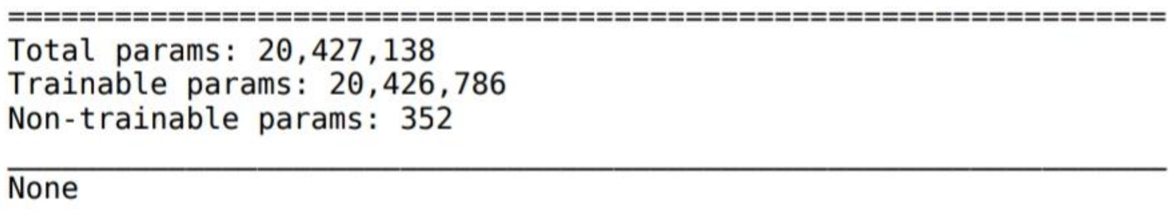


Fig 2



```
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 100, 100, 3)]     0

 conv2d_35 (Conv2D)          (None, 23, 23, 48)        17472

 batch_normalization_14 (Bat  (None, 23, 23, 48)       192
 chNormalization)

 tf.nn.relu_42 (TFOpLambda)  (None, 23, 23, 48)        0

 max_pooling2d_21 (MaxPoolin  (None, 11, 11, 48)       0
 g2D)

 conv2d_36 (Conv2D)          (None, 11, 11, 128)       743552

 batch_normalization_15 (Bat  (None, 11, 11, 128)      512
 chNormalization)

 tf.nn.relu_43 (TFOpLambda)  (None, 11, 11, 128)       0

 max_pooling2d_22 (MaxPoolin  (None, 5, 5, 128)        0
 g2D)

 conv2d_37 (Conv2D)          (None, 5, 5, 192)         221376

 tf.nn.relu_44 (TFOpLambda)  (None, 5, 5, 192)         0

 conv2d_38 (Conv2D)          (None, 5, 5, 192)         331968

 tf.nn.relu_45 (TFOpLambda)  (None, 5, 5, 192)         0

 conv2d_39 (Conv2D)          (None, 5, 5, 128)         221312

 tf.nn.relu_46 (TFOpLambda)  (None, 5, 5, 128)         0

 max_pooling2d_23 (MaxPoolin  (None, 2, 2, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 512)               0

 dense_21 (Dense)            (None, 4096)              2101248

 dense_22 (Dense)            (None, 4096)              16781312

 dense_23 (Dense)            (None, 2)                 8194

 tf.nn.relu_47 (TFOpLambda)  (None, 2)                 0
```

Fig 3



```
=================================================================
Total params: 20,427,138
Trainable params: 20,426,786
Non-trainable params: 352
_________________________________________________________________
None
```

Fig 4

3.1.6 Training model

For this process, we trained the model for 100 epochs for each joint using the lsp dataset different joints show different accuracy while training the list of final accuracy for each training set are as follows:

Joints	Accuracy
Right Knee	96.86%
Right ankle	98.33%
Right hip	97%
Left hip	98.14%
Left Knee	97.14%
Left ankle	99.14%
Right Wrist	97.86%
Right elbow	96.79%
Right shoulder	98.21%
Left shoulder	98.79%
Left elbow	97.57%
Left wrist	98.50%
Neck	99.29%
Head	99.64%

Table 1

3.1.7 Evaluation

The data set is tested by using 30% of the original LSP dataset, then we evaluated it for how well it performs in unseen data. The result are as follows.

Joints	Accuracy
Right Knee	86.83%
Right ankle	92.33%
Right hip	73.17%
Left hip	70.00%
Left Knee	80.50%
Left ankle	91.00%
Right Wrist	85.67%
Right elbow	85.50%
Right shoulder	86.21%
Left shoulder	88.83%
Left elbow	76.50%
Left wrist	73.17%
Neck	94.33%
Head	95.00%

Table 2

3.1.8 Saving the models

After training the model we saved the model using pickle module. For each joint we created a .sav file containing the weights and biases of the neurons of the model and it can be loaded at anytime in the future to predict the coordinates of the joints in any image containing a human figure.

3.1.9 Constructing the final skeleton

After we saved all the weights and biases for each joint we load all the weights and biases in different models for different joints. Then use those models to predict all the joints and then connect the adjacent joints to construct the human skeleton to estimate the final pose of the human body.

3.2 Data Annotation Tool

3.2.1 introduction

In machine learning, data annotation is the process of labelling data to show the outcome you want your machine learning model to predict. We are marking - labelling, tagging, transcribing, or processing - a dataset with the features we want our machine learning system to learn to recognize. Once our model is deployed, we want it to recognize those features on its own and make a decision or take some action as a result.

3.2.2 working

The dataset we used to train the model i.e lsp or leeds sports pose was very low resolution and poses were very extreme instead of being subtle so we decided to build a data annotation tool that can take images of people posing and annotate 14 joints in that pose with the help of an user it is a simple tool which uses web technologies such as html , css , javascript and node to load images from a back-end server to a front-end page and this page has 14 different markers which are color-coded according to the joints and these markers are all draggable these draggable markers can be dragged to the right position of the joints and when all the joints fall into the right positions we can save the coordinate and download a file containing the coordinates of the joints in the image. The tool writes in a .csv format file which then can be read using python as a numpy array. We then used this numpy array and the image in our previously built model to train/test using our previously built CNN models.

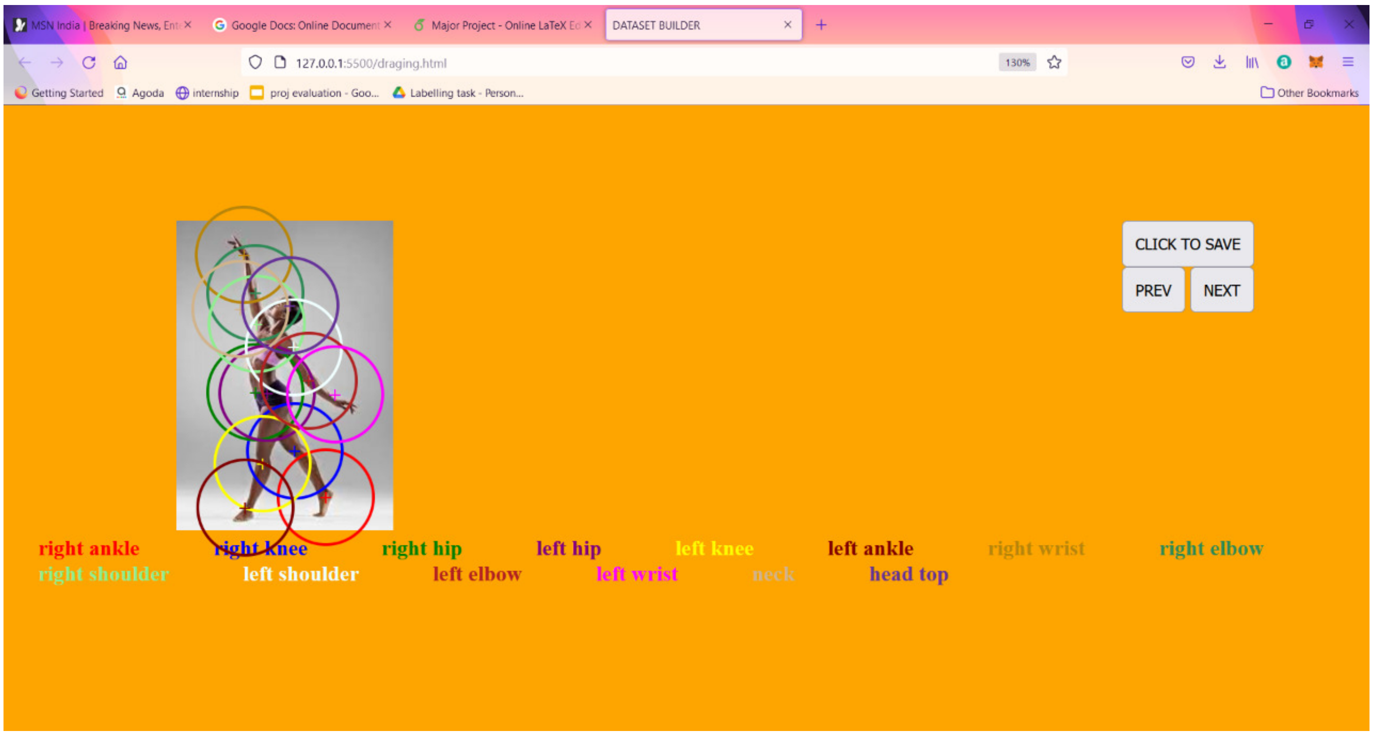


Fig 5

3.3 Comparing with other CNN models

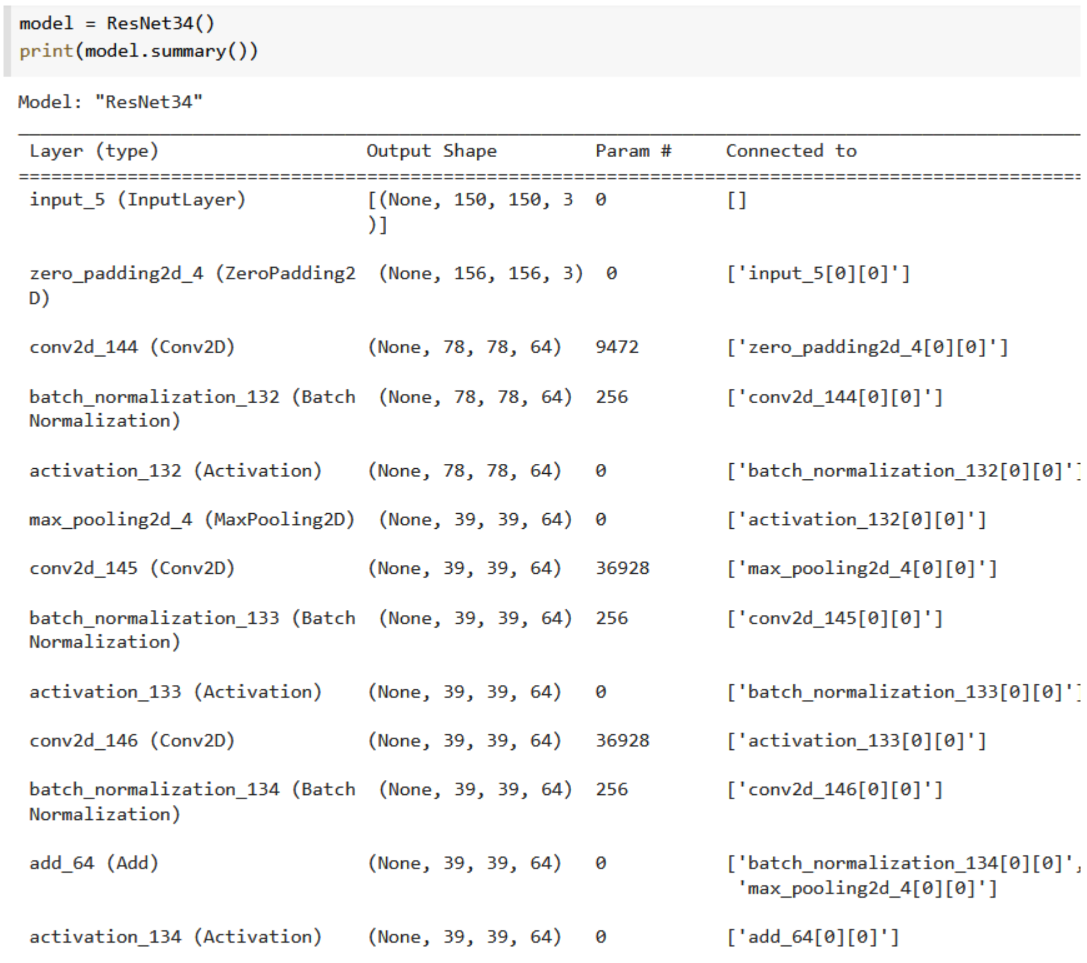
3.3.1 introduction

After implementing the deep pose model with our own modifications and adjustments. We decided to replace the CNN model with some of the newer architectures available today like AlexNet and ResNet34 to see how well they perform in this problem and benchmark them with our own implementation

3.3.2 ResNet34

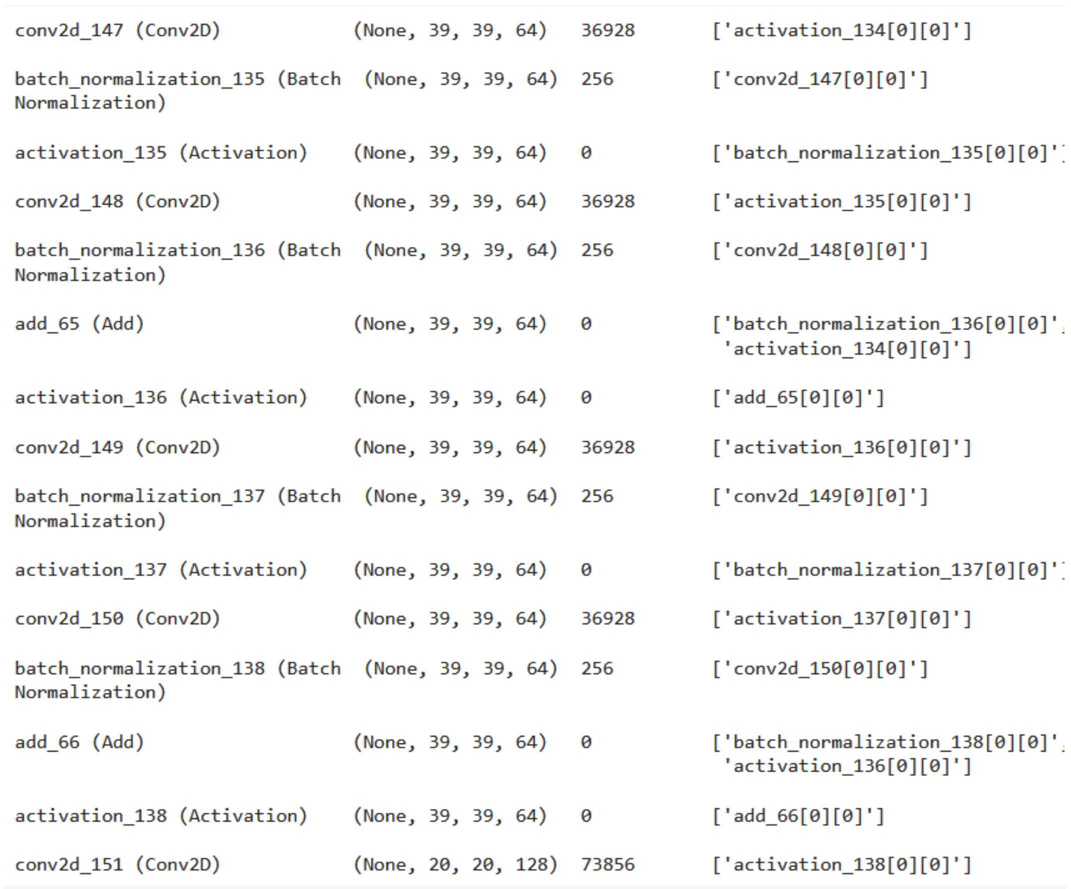
Implementing the ResNet 35 architecture

ResNet, short for Residual Network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper "Deep Residual Learning for Image Recognition". Mostly in order to solve a complex problem, we stack some additional layers in the Deep Neural Networks which results in improved accuracy and performance. The intuition behind adding more layers is that these layers progressively learn more complex features. For example, in case of recognising images, the first layer may learn to detect edges, the second layer may learn to identify textures and similarly the third layer can learn to detect objects and so on. But it has been found that there is a maximum threshold for depth with the traditional Convolutional neural network model Residual networks have become quite popular for image recognition and classification tasks because of their ability to solve vanishing and exploding gradients when adding more layers to an already deep neural network. We replaced the model with ResNet 35 architectural constraints by changing the underlying layers and its parameters.

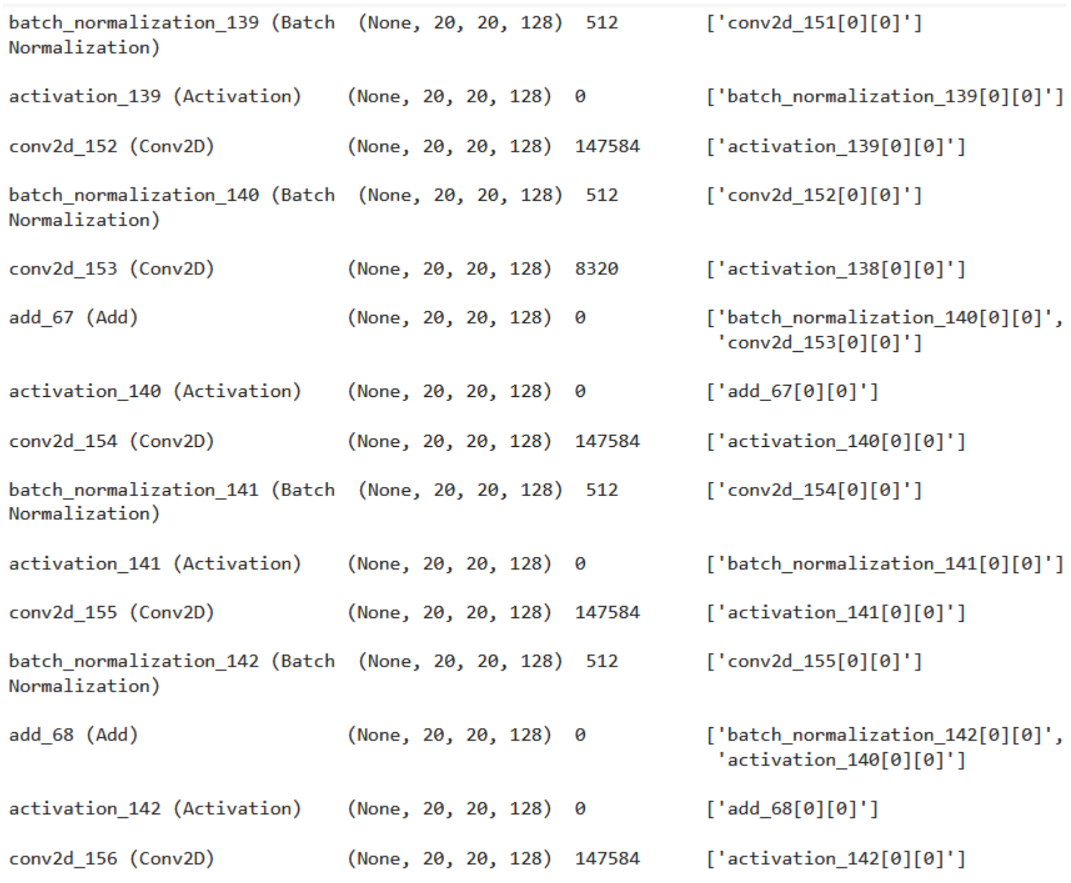


```
model = ResNet34()
print(model.summary())
```

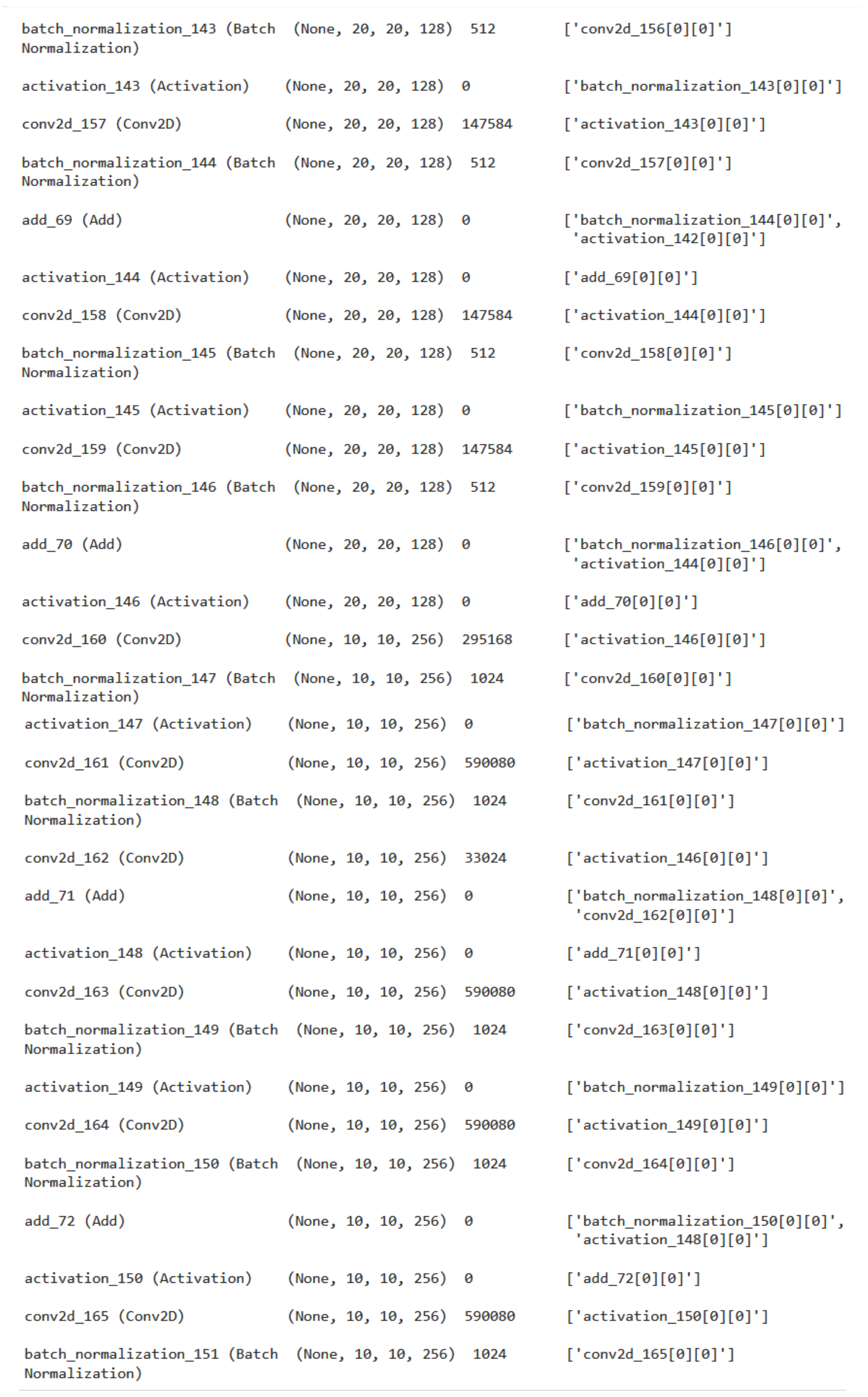
```
Model: "ResNet34"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_5 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 zero_padding2d_4 (ZeroPadding2  (None, 156, 156, 3)  0          ['input_5[0][0]']
 D)

 conv2d_144 (Conv2D)            (None, 78, 78, 64)   9472        ['zero_padding2d_4[0][0]']

 batch_normalization_132 (Batch  (None, 78, 78, 64)  256         ['conv2d_144[0][0]']
 Normalization)

 activation_132 (Activation)    (None, 78, 78, 64)   0           ['batch_normalization_132[0][0]'

 max_pooling2d_4 (MaxPooling2D)  (None, 39, 39, 64)  0           ['activation_132[0][0]']

 conv2d_145 (Conv2D)            (None, 39, 39, 64)   36928       ['max_pooling2d_4[0][0]']

 batch_normalization_133 (Batch  (None, 39, 39, 64)  256         ['conv2d_145[0][0]']
 Normalization)

 activation_133 (Activation)    (None, 39, 39, 64)   0           ['batch_normalization_133[0][0]'

 conv2d_146 (Conv2D)            (None, 39, 39, 64)   36928       ['activation_133[0][0]']

 batch_normalization_134 (Batch  (None, 39, 39, 64)  256         ['conv2d_146[0][0]']
 Normalization)

 add_64 (Add)                   (None, 39, 39, 64)   0           ['batch_normalization_134[0][0]',
                                                                  'max_pooling2d_4[0][0]']

 activation_134 (Activation)    (None, 39, 39, 64)   0           ['add_64[0][0]']
```



```
conv2d_147 (Conv2D)            (None, 39, 39, 64)  36928       ['activation_134[0][0]']

batch_normalization_135 (Batch  (None, 39, 39, 64)  256        ['conv2d_147[0][0]']
Normalization)

activation_135 (Activation)    (None, 39, 39, 64)  0           ['batch_normalization_135[0][0]'

conv2d_148 (Conv2D)            (None, 39, 39, 64)  36928       ['activation_135[0][0]']

batch_normalization_136 (Batch  (None, 39, 39, 64)  256        ['conv2d_148[0][0]']
Normalization)

add_65 (Add)                   (None, 39, 39, 64)  0           ['batch_normalization_136[0][0]',
                                                                'activation_134[0][0]']

activation_136 (Activation)    (None, 39, 39, 64)  0           ['add_65[0][0]']

conv2d_149 (Conv2D)            (None, 39, 39, 64)  36928       ['activation_136[0][0]']

batch_normalization_137 (Batch  (None, 39, 39, 64)  256        ['conv2d_149[0][0]']
Normalization)

activation_137 (Activation)    (None, 39, 39, 64)  0           ['batch_normalization_137[0][0]'

conv2d_150 (Conv2D)            (None, 39, 39, 64)  36928       ['activation_137[0][0]']

batch_normalization_138 (Batch  (None, 39, 39, 64)  256        ['conv2d_150[0][0]']
Normalization)

add_66 (Add)                   (None, 39, 39, 64)  0           ['batch_normalization_138[0][0]',
                                                                'activation_136[0][0]']

activation_138 (Activation)    (None, 39, 39, 64)  0           ['add_66[0][0]']

conv2d_151 (Conv2D)            (None, 20, 20, 128)  73856      ['activation_138[0][0]']
```

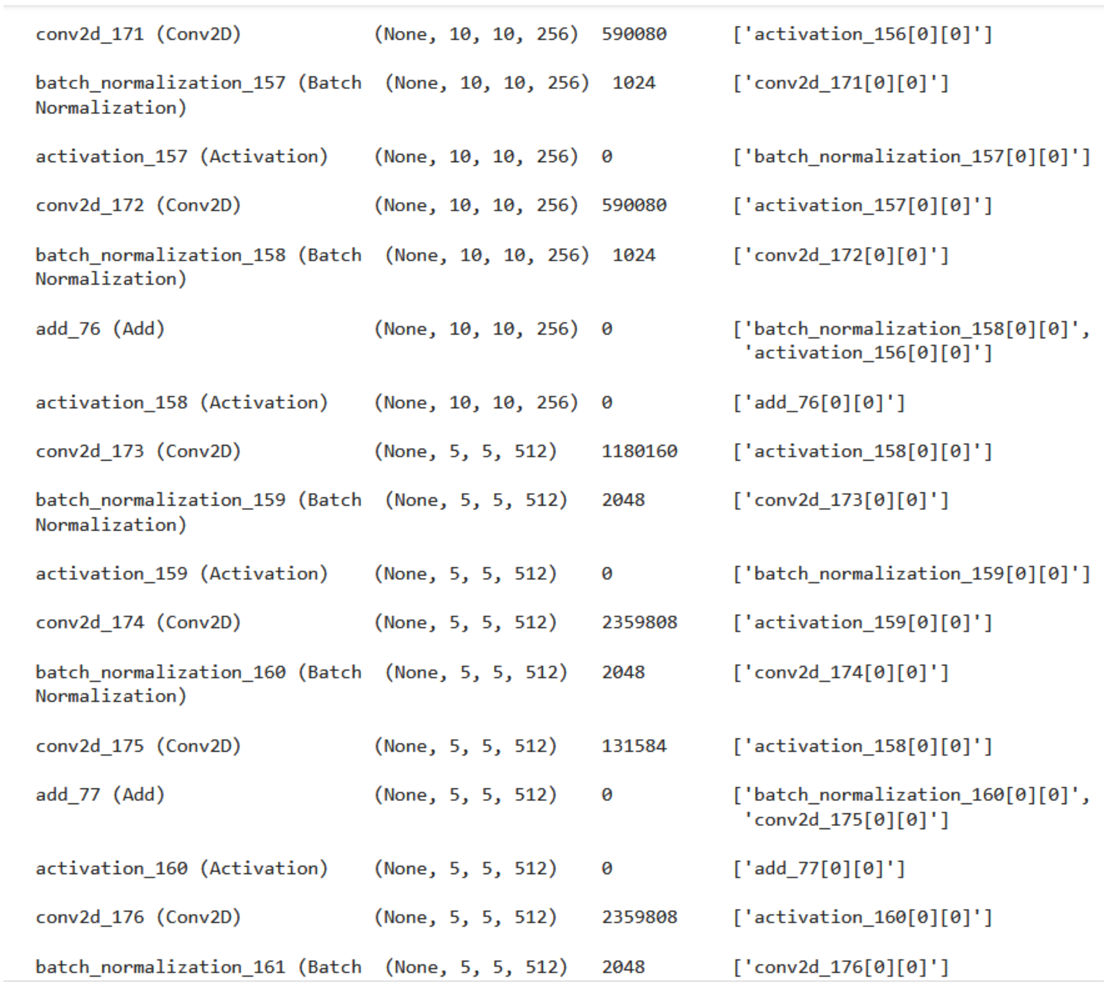


```
batch_normalization_139 (Batch  (None, 20, 20, 128)  512       ['conv2d_151[0][0]']
Normalization)

activation_139 (Activation)    (None, 20, 20, 128)  0          ['batch_normalization_139[0][0]']

conv2d_152 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_139[0][0]']

batch_normalization_140 (Batch  (None, 20, 20, 128)  512       ['conv2d_152[0][0]']
Normalization)

conv2d_153 (Conv2D)            (None, 20, 20, 128)  8320       ['activation_138[0][0]']

add_67 (Add)                   (None, 20, 20, 128)  0          ['batch_normalization_140[0][0]',
                                                                'conv2d_153[0][0]']

activation_140 (Activation)    (None, 20, 20, 128)  0          ['add_67[0][0]']

conv2d_154 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_140[0][0]']

batch_normalization_141 (Batch  (None, 20, 20, 128)  512       ['conv2d_154[0][0]']
Normalization)

activation_141 (Activation)    (None, 20, 20, 128)  0          ['batch_normalization_141[0][0]']

conv2d_155 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_141[0][0]']

batch_normalization_142 (Batch  (None, 20, 20, 128)  512       ['conv2d_155[0][0]']
Normalization)

add_68 (Add)                   (None, 20, 20, 128)  0          ['batch_normalization_142[0][0]',
                                                                'activation_140[0][0]']

activation_142 (Activation)    (None, 20, 20, 128)  0          ['add_68[0][0]']

conv2d_156 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_142[0][0]']
```

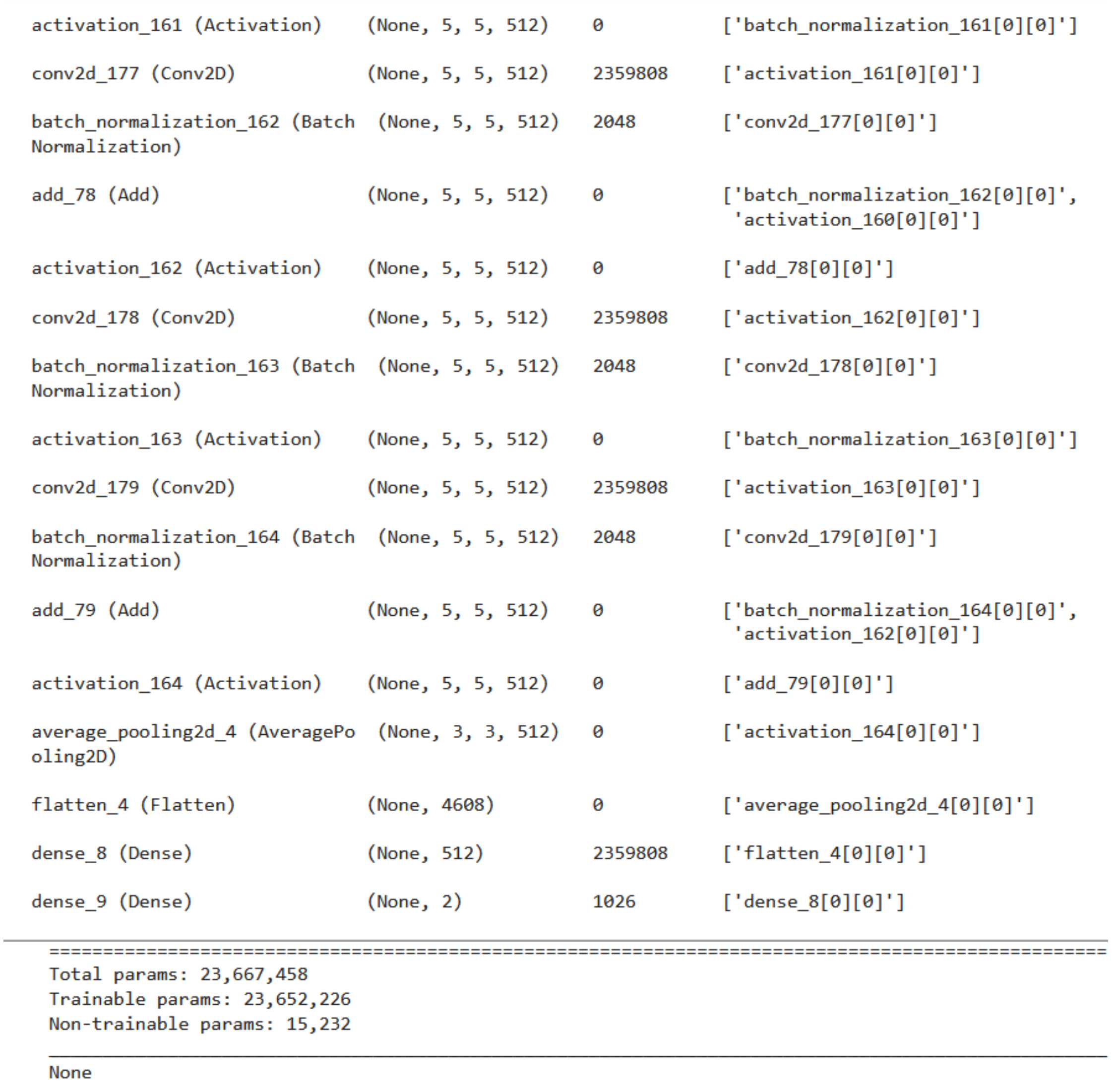


```
 batch_normalization_143 (Batch  (None, 20, 20, 128)  512        ['conv2d_156[0][0]']
 Normalization)

 activation_143 (Activation)    (None, 20, 20, 128)  0          ['batch_normalization_143[0][0]']

 conv2d_157 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_143[0][0]']

 batch_normalization_144 (Batch  (None, 20, 20, 128)  512        ['conv2d_157[0][0]']
 Normalization)

 add_69 (Add)                   (None, 20, 20, 128)  0          ['batch_normalization_144[0][0]',
                                                                  'activation_142[0][0]']

 activation_144 (Activation)    (None, 20, 20, 128)  0          ['add_69[0][0]']

 conv2d_158 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_144[0][0]']

 batch_normalization_145 (Batch  (None, 20, 20, 128)  512        ['conv2d_158[0][0]']
 Normalization)

 activation_145 (Activation)    (None, 20, 20, 128)  0          ['batch_normalization_145[0][0]']

 conv2d_159 (Conv2D)            (None, 20, 20, 128)  147584     ['activation_145[0][0]']

 batch_normalization_146 (Batch  (None, 20, 20, 128)  512        ['conv2d_159[0][0]']
 Normalization)

 add_70 (Add)                   (None, 20, 20, 128)  0          ['batch_normalization_146[0][0]',
                                                                  'activation_144[0][0]']

 activation_146 (Activation)    (None, 20, 20, 128)  0          ['add_70[0][0]']

 conv2d_160 (Conv2D)            (None, 10, 10, 256)  295168     ['activation_146[0][0]']

 batch_normalization_147 (Batch  (None, 10, 10, 256)  1024       ['conv2d_160[0][0]']
 Normalization)
 activation_147 (Activation)    (None, 10, 10, 256)  0          ['batch_normalization_147[0][0]']

 conv2d_161 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_147[0][0]']

 batch_normalization_148 (Batch  (None, 10, 10, 256)  1024       ['conv2d_161[0][0]']
 Normalization)

 conv2d_162 (Conv2D)            (None, 10, 10, 256)  33024      ['activation_146[0][0]']

 add_71 (Add)                   (None, 10, 10, 256)  0          ['batch_normalization_148[0][0]',
                                                                  'conv2d_162[0][0]']

 activation_148 (Activation)    (None, 10, 10, 256)  0          ['add_71[0][0]']

 conv2d_163 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_148[0][0]']

 batch_normalization_149 (Batch  (None, 10, 10, 256)  1024       ['conv2d_163[0][0]']
 Normalization)

 activation_149 (Activation)    (None, 10, 10, 256)  0          ['batch_normalization_149[0][0]']

 conv2d_164 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_149[0][0]']

 batch_normalization_150 (Batch  (None, 10, 10, 256)  1024       ['conv2d_164[0][0]']
 Normalization)

 add_72 (Add)                   (None, 10, 10, 256)  0          ['batch_normalization_150[0][0]',
                                                                  'activation_148[0][0]']

 activation_150 (Activation)    (None, 10, 10, 256)  0          ['add_72[0][0]']

 conv2d_165 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_150[0][0]']

 batch_normalization_151 (Batch  (None, 10, 10, 256)  1024       ['conv2d_165[0][0]']
 Normalization)
```

```
activation_151 (Activation)    (None, 10, 10, 256)  0          ['batch_normalization_151[0][0]']

conv2d_166 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_151[0][0]']

batch_normalization_152 (Batch  (None, 10, 10, 256)  1024       ['conv2d_166[0][0]']
Normalization)

add_73 (Add)                   (None, 10, 10, 256)  0          ['batch_normalization_152[0][0]',
                                                                 'activation_150[0][0]']

activation_152 (Activation)    (None, 10, 10, 256)  0          ['add_73[0][0]']

conv2d_167 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_152[0][0]']

batch_normalization_153 (Batch  (None, 10, 10, 256)  1024       ['conv2d_167[0][0]']
Normalization)

activation_153 (Activation)    (None, 10, 10, 256)  0          ['batch_normalization_153[0][0]']

conv2d_168 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_153[0][0]']

batch_normalization_154 (Batch  (None, 10, 10, 256)  1024       ['conv2d_168[0][0]']
Normalization)

add_74 (Add)                   (None, 10, 10, 256)  0          ['batch_normalization_154[0][0]',
                                                                 'activation_152[0][0]']

activation_154 (Activation)    (None, 10, 10, 256)  0          ['add_74[0][0]']

conv2d_169 (Conv2D)            (None, 10, 10, 256)  590080     ['activation_154[0][0]']

batch_normalization_155 (Batch  (None, 10, 10, 256)  1024       ['conv2d_169[0][0]']
Normalization)

activation_155 (Activation)    (None, 10, 10, 256)  0          ['batch_normalization_155[0][0]']
```



```
 conv2d_171 (Conv2D)            (None, 10, 10, 256)  590080      ['activation_156[0][0]']

 batch_normalization_157 (Batch  (None, 10, 10, 256)  1024       ['conv2d_171[0][0]']
 Normalization)

 activation_157 (Activation)    (None, 10, 10, 256)  0           ['batch_normalization_157[0][0]']

 conv2d_172 (Conv2D)            (None, 10, 10, 256)  590080      ['activation_157[0][0]']

 batch_normalization_158 (Batch  (None, 10, 10, 256)  1024       ['conv2d_172[0][0]']
 Normalization)

 add_76 (Add)                   (None, 10, 10, 256)  0           ['batch_normalization_158[0][0]',
                                                                  'activation_156[0][0]']

 activation_158 (Activation)    (None, 10, 10, 256)  0           ['add_76[0][0]']

 conv2d_173 (Conv2D)            (None, 5, 5, 512)    1180160     ['activation_158[0][0]']

 batch_normalization_159 (Batch  (None, 5, 5, 512)   2048        ['conv2d_173[0][0]']
 Normalization)

 activation_159 (Activation)    (None, 5, 5, 512)    0           ['batch_normalization_159[0][0]']

 conv2d_174 (Conv2D)            (None, 5, 5, 512)    2359808     ['activation_159[0][0]']

 batch_normalization_160 (Batch  (None, 5, 5, 512)   2048        ['conv2d_174[0][0]']
 Normalization)

 conv2d_175 (Conv2D)            (None, 5, 5, 512)    131584      ['activation_158[0][0]']

 add_77 (Add)                   (None, 5, 5, 512)    0           ['batch_normalization_160[0][0]',
                                                                  'conv2d_175[0][0]']

 activation_160 (Activation)    (None, 5, 5, 512)    0           ['add_77[0][0]']

 conv2d_176 (Conv2D)            (None, 5, 5, 512)    2359808     ['activation_160[0][0]']

 batch_normalization_161 (Batch  (None, 5, 5, 512)   2048        ['conv2d_176[0][0]']
```



```
activation_161 (Activation)    (None, 5, 5, 512)    0          ['batch_normalization_161[0][0]']

conv2d_177 (Conv2D)            (None, 5, 5, 512)    2359808    ['activation_161[0][0]']

batch_normalization_162 (Batch  (None, 5, 5, 512)   2048       ['conv2d_177[0][0]']
Normalization)

add_78 (Add)                   (None, 5, 5, 512)    0          ['batch_normalization_162[0][0]',
                                                                 'activation_160[0][0]']

activation_162 (Activation)    (None, 5, 5, 512)    0          ['add_78[0][0]']

conv2d_178 (Conv2D)            (None, 5, 5, 512)    2359808    ['activation_162[0][0]']

batch_normalization_163 (Batch  (None, 5, 5, 512)   2048       ['conv2d_178[0][0]']
Normalization)

activation_163 (Activation)    (None, 5, 5, 512)    0          ['batch_normalization_163[0][0]']

conv2d_179 (Conv2D)            (None, 5, 5, 512)    2359808    ['activation_163[0][0]']

batch_normalization_164 (Batch  (None, 5, 5, 512)   2048       ['conv2d_179[0][0]']
Normalization)

add_79 (Add)                   (None, 5, 5, 512)    0          ['batch_normalization_164[0][0]',
                                                                 'activation_162[0][0]']

activation_164 (Activation)    (None, 5, 5, 512)    0          ['add_79[0][0]']

average_pooling2d_4 (AveragePo  (None, 3, 3, 512)   0          ['activation_164[0][0]']
oling2D)

flatten_4 (Flatten)            (None, 4608)         0          ['average_pooling2d_4[0][0]']

dense_8 (Dense)                (None, 512)          2359808    ['flatten_4[0][0]']

dense_9 (Dense)                (None, 2)            1026       ['dense_8[0][0]']

==================================================================================================
Total params: 23,667,458
Trainable params: 23,652,226
Non-trainable params: 15,232
__________________________________________________________________________________________________
None
```

Fig 6

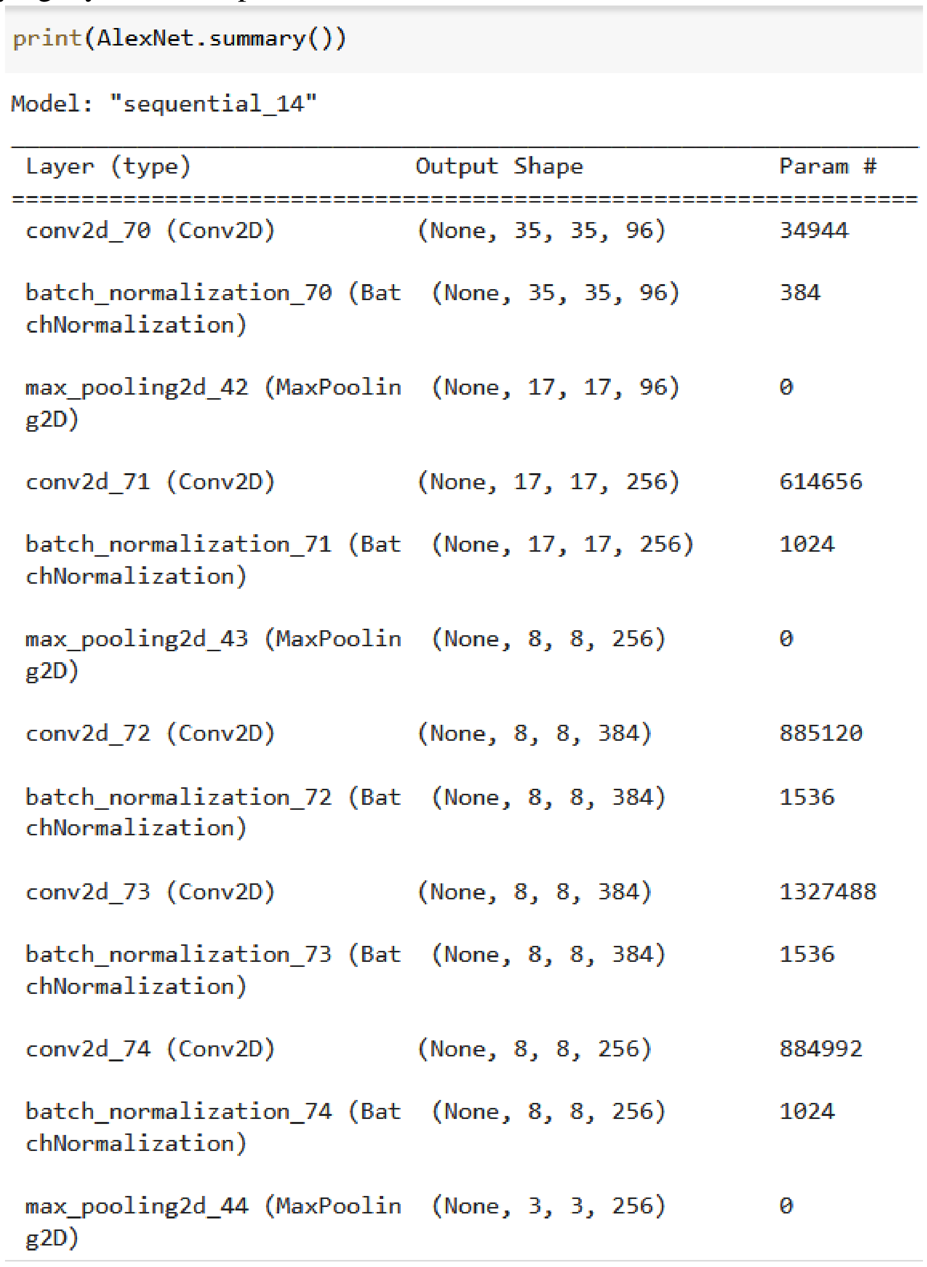
3.3.3 AlexNet

Implementing the AlexNet architecture

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor. In AlexNet's first layer, the convolution window shape is 11×11. Since most images in ImageNet are more than ten times higher and wider than the MNIST images, objects in ImageNet data tend to occupy more pixels. Consequently, a larger convolution window is needed to capture the object. The convolution window shape in the second layer is reduced to 5×5, followed by

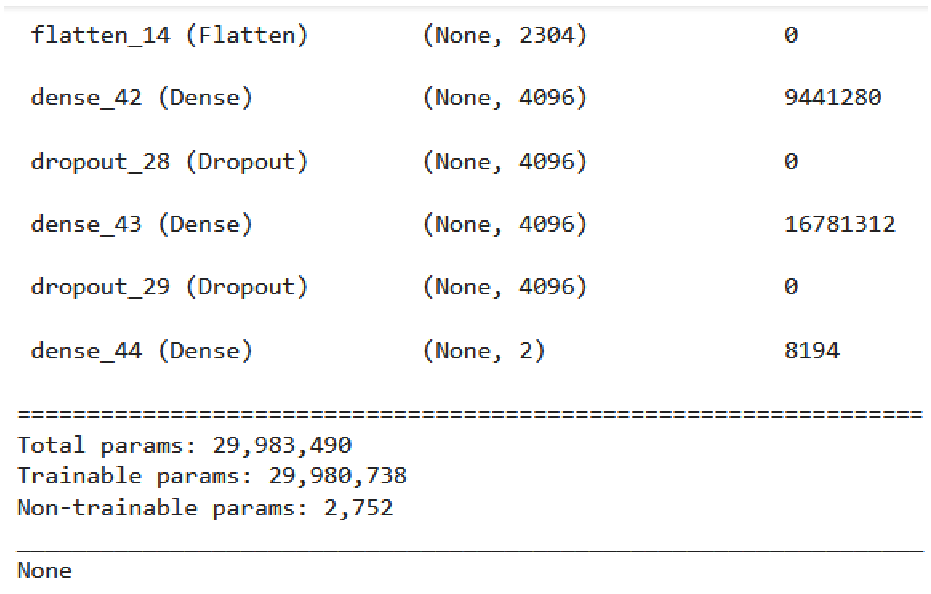
3×3. In addition, after the first, second, and fifth convolutional layers, the network adds maximum pooling layers with a window shape of 3×3 and a stride of 2. Moreover, AlexNet has ten times more convolution channels than LeNet. After the last convolutional layer there are two fully-connected layers with 4096 outputs. These two huge fully-connected layers produce model parameters of nearly 1 GB. AlexNet uses the ReLU activation function.

We replaced the model with AlexNet architectural constraints by changing the underlying layers and its parameters.



```
print(AlexNet.summary())
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_70 (Conv2D)          (None, 35, 35, 96)        34944     
                                                                 
 batch_normalization_70 (Bat  (None, 35, 35, 96)       384       
 chNormalization)                                                
                                                                 
 max_pooling2d_42 (MaxPoolin  (None, 17, 17, 96)       0         
 g2D)                                                            
                                                                 
 conv2d_71 (Conv2D)          (None, 17, 17, 256)       614656    
                                                                 
 batch_normalization_71 (Bat  (None, 17, 17, 256)      1024      
 chNormalization)                                                
                                                                 
 max_pooling2d_43 (MaxPoolin  (None, 8, 8, 256)        0         
 g2D)                                                            
                                                                 
 conv2d_72 (Conv2D)          (None, 8, 8, 384)         885120    
                                                                 
 batch_normalization_72 (Bat  (None, 8, 8, 384)        1536      
 chNormalization)                                                
                                                                 
 conv2d_73 (Conv2D)          (None, 8, 8, 384)         1327488   
                                                                 
 batch_normalization_73 (Bat  (None, 8, 8, 384)        1536      
 chNormalization)                                                
                                                                 
 conv2d_74 (Conv2D)          (None, 8, 8, 256)         884992    
                                                                 
 batch_normalization_74 (Bat  (None, 8, 8, 256)        1024      
 chNormalization)                                                
                                                                 
 max_pooling2d_44 (MaxPoolin  (None, 3, 3, 256)        0         
 g2D)                                                            
```



```
 flatten_14 (Flatten)         (None, 2304)             0

 dense_42 (Dense)             (None, 4096)             9441280

 dropout_28 (Dropout)         (None, 4096)             0

 dense_43 (Dense)             (None, 4096)             16781312

 dropout_29 (Dropout)         (None, 4096)             0

 dense_44 (Dense)             (None, 2)                8194

=================================================================
Total params: 29,983,490
Trainable params: 29,980,738
Non-trainable params: 2,752
_________________________________________________________________
None
```

Fig 7

Chapter 4

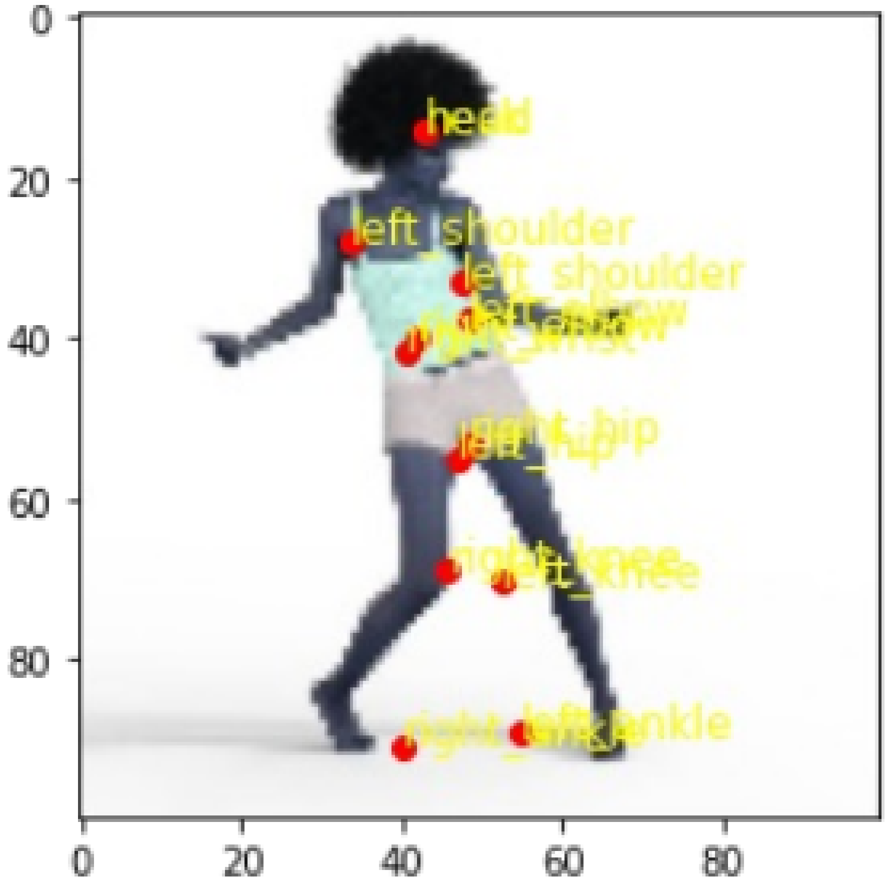
Results and Discussion

4.1 Results

After running our model we, get the final result as shown below.

Joints	Accuracy
Right Knee	86.83%
Right ankle	92.33%
Right hip	73.17%
Left hip	70.00%
Left Knee	80.50%
Left ankle	91.00%
Right Wrist	85.67%
Right elbow	85.50%
Right shoulder	86.21%
Left shoulder	88.83%
Left elbow	76.50%
Left wrist	73.17%
Neck	94.33%
Head	95.00%

Table 3



head
left_shoulder
left_shoulder
left_elbow
right_wrist
right_hip
left_hip
right_knee
left_knee
right_ankle
left_ankle
0
20
40
60
80
0
20
40
60
80

Fig 8

4.2 Accuracy

The accuracy is as follows:

Left hip

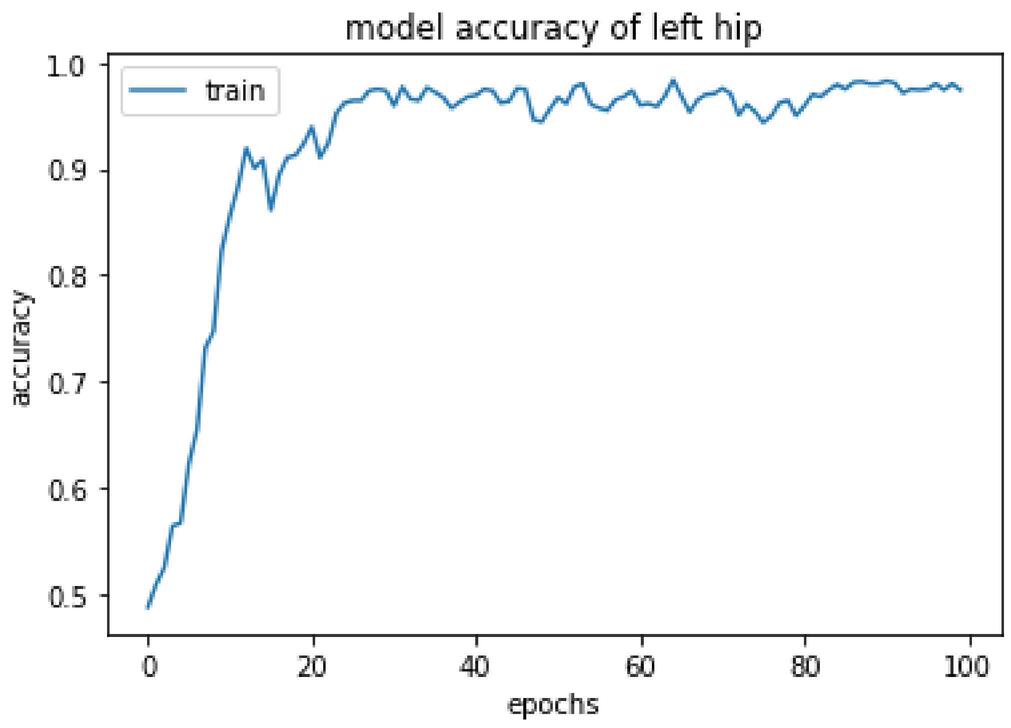


Fig 9

Left Knee

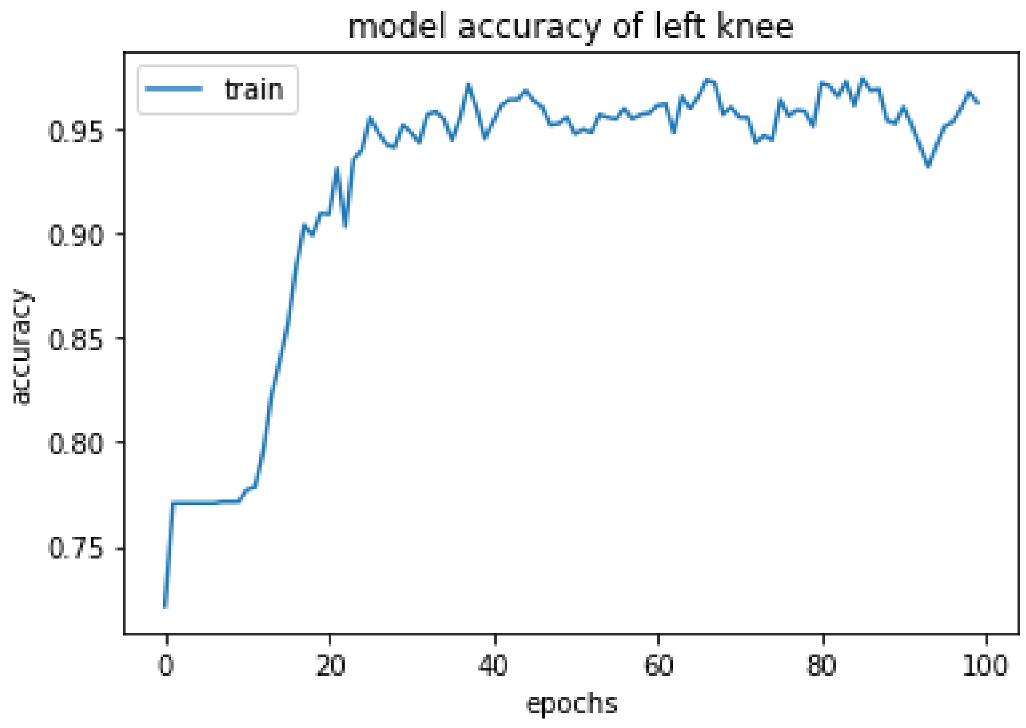


Fig 10

Left ankle

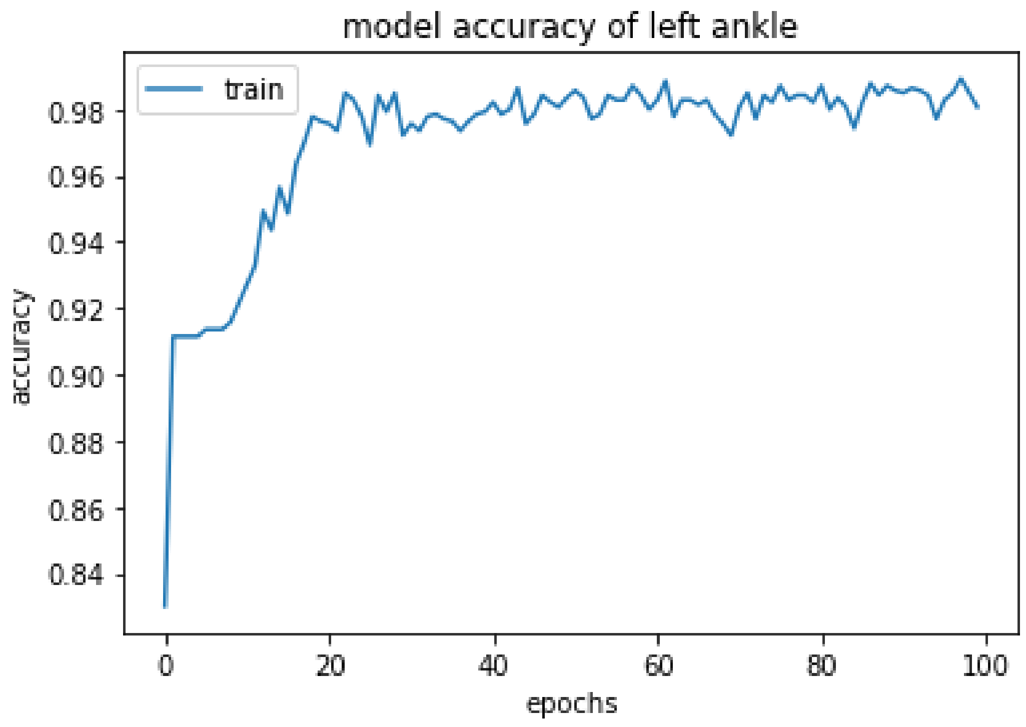


Fig 11

Left shoulder

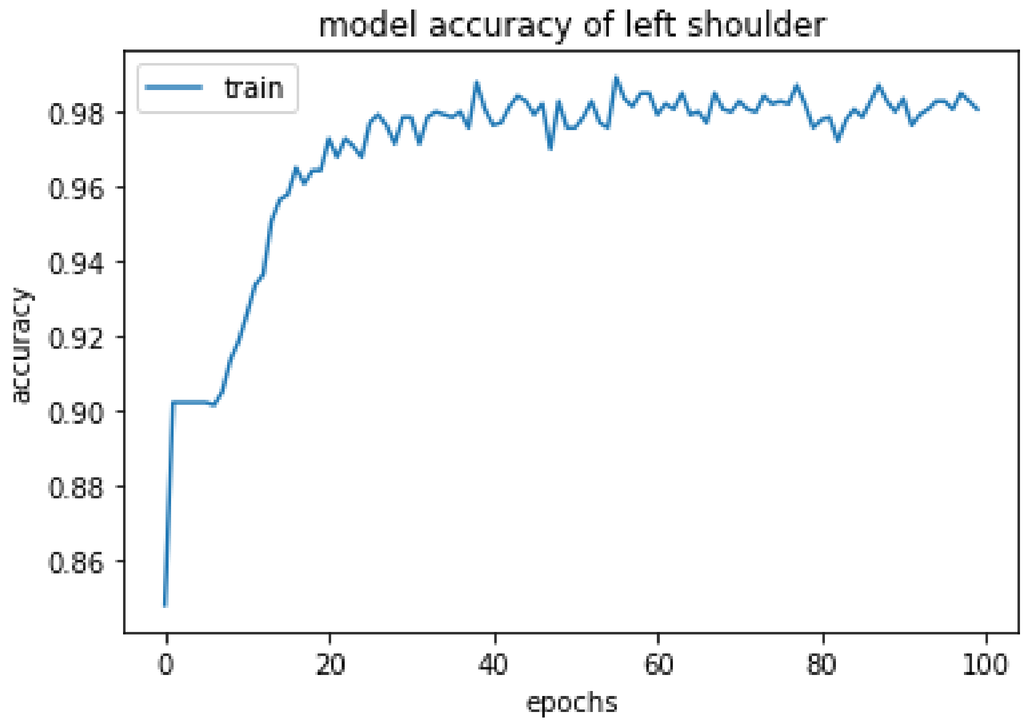


Fig 12

Left elbow

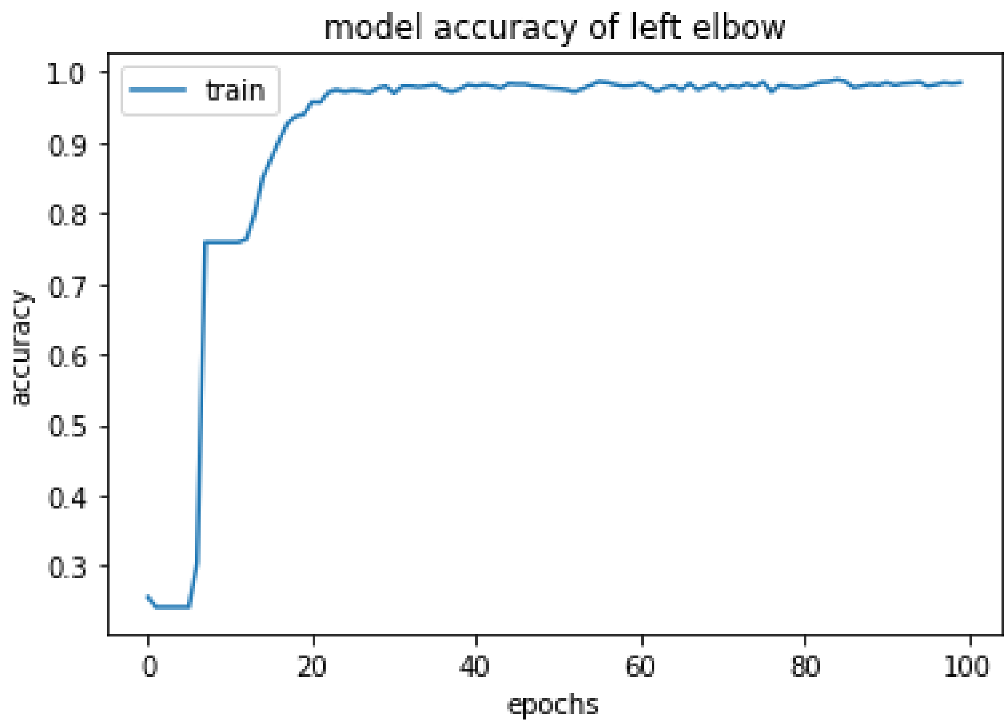


Fig 13

Left wrist

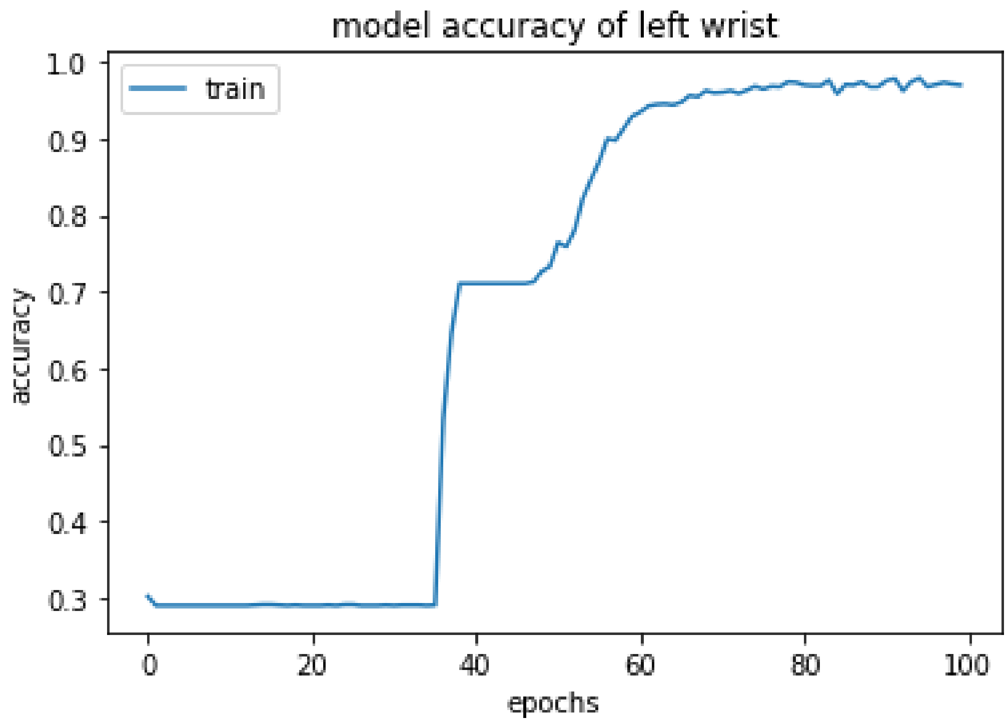


Fig 14

Right Wrist

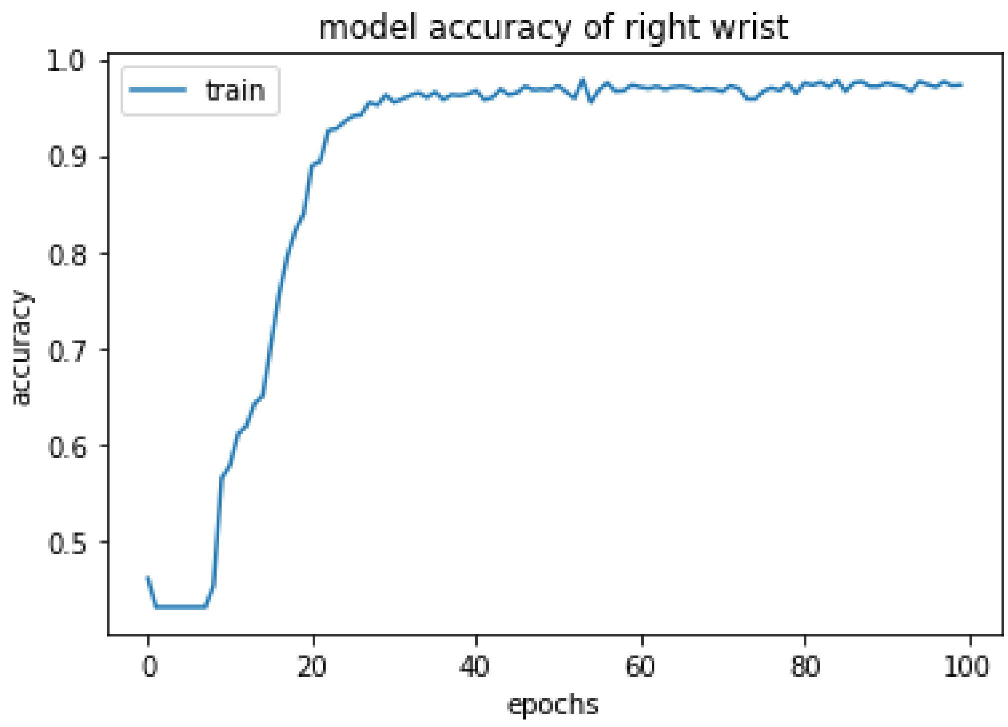


Fig 15

Right elbow

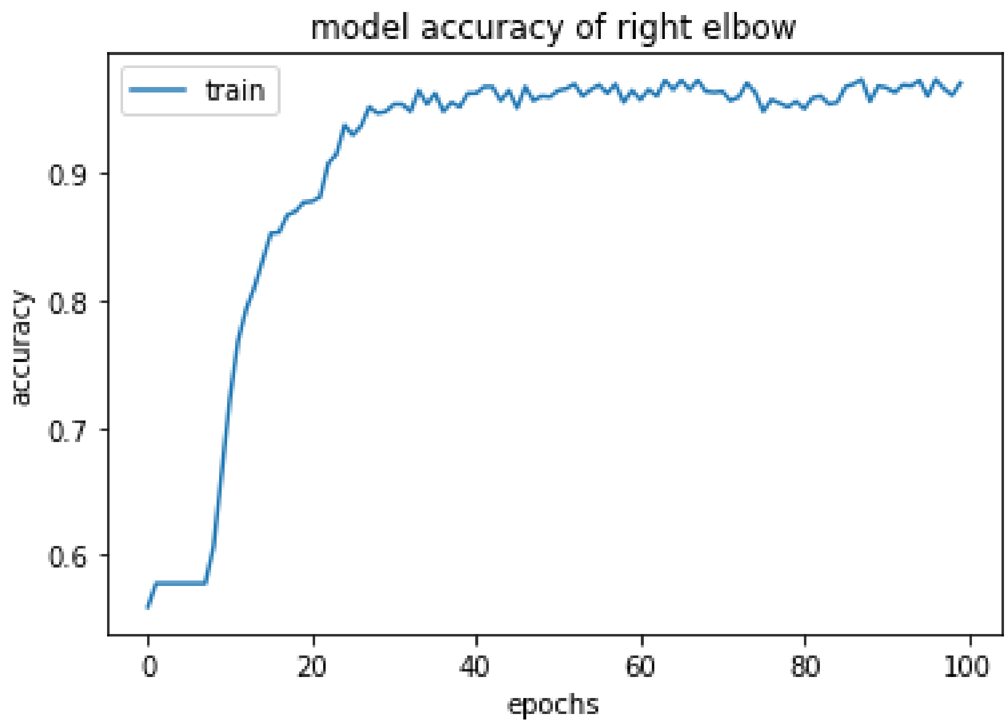


Fig 16

Right shoulder

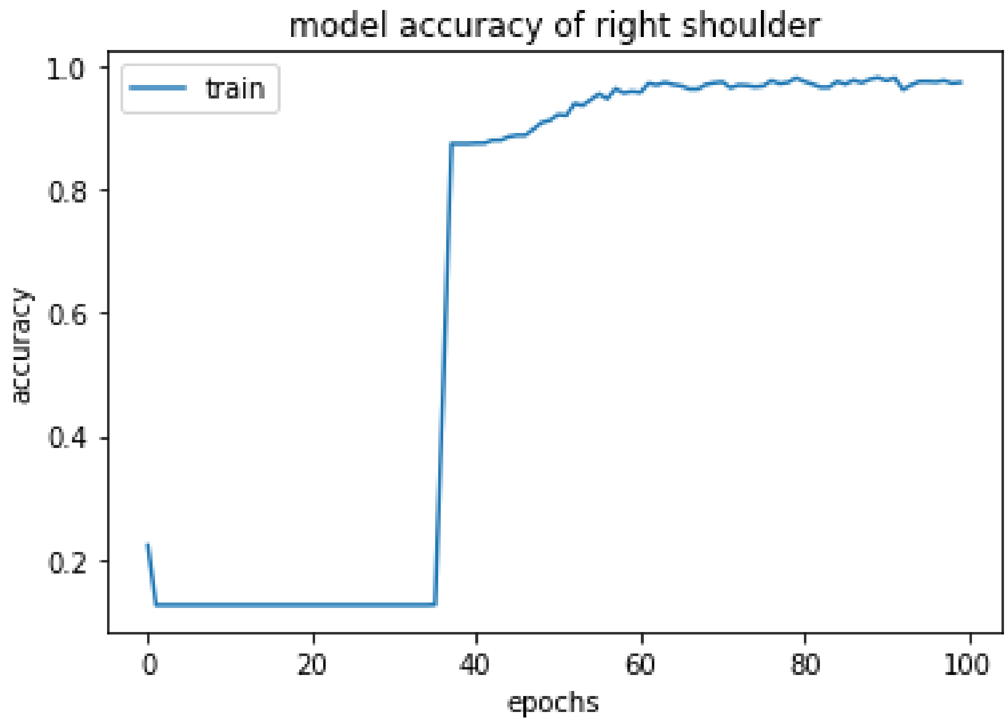


Fig 17

Right Knee

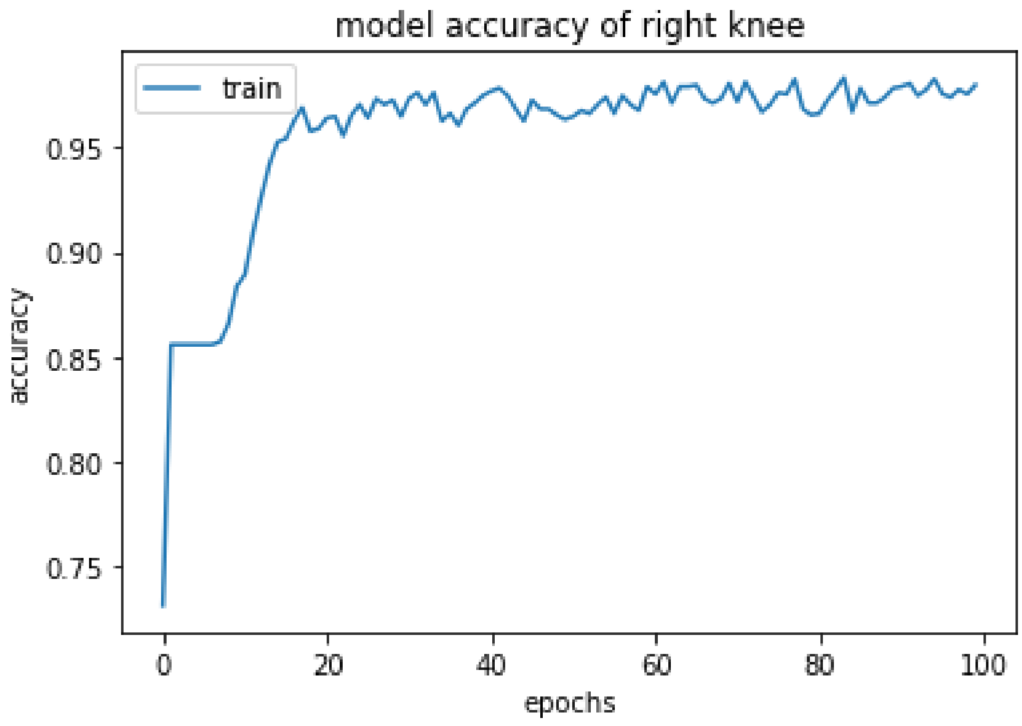


Fig 18

Right ankle

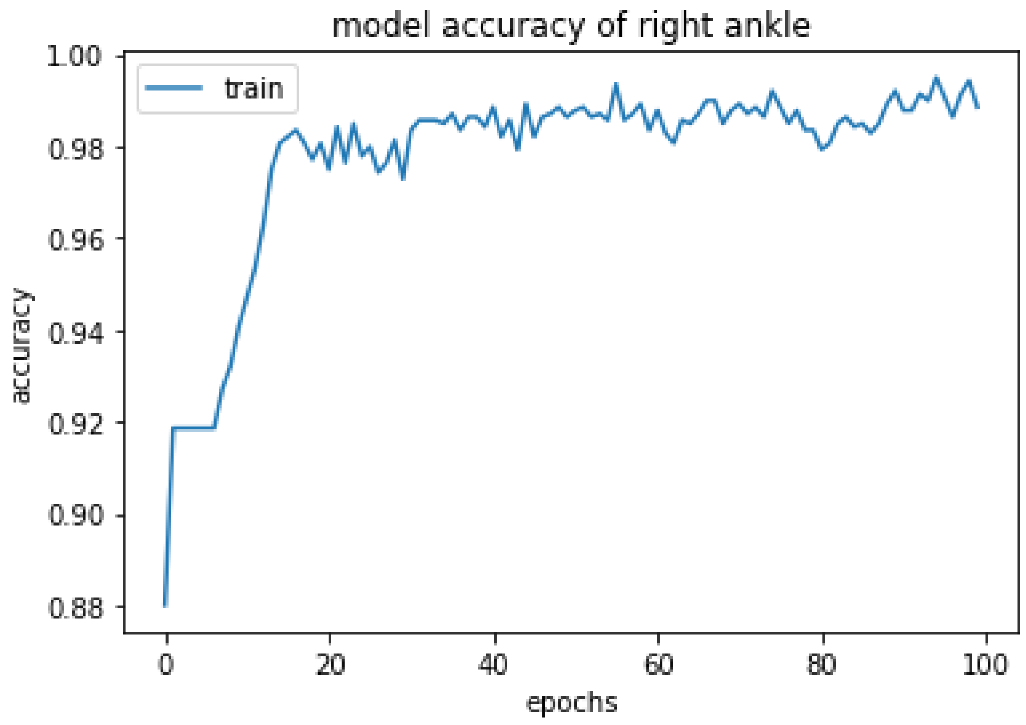


Fig 19

Right hip

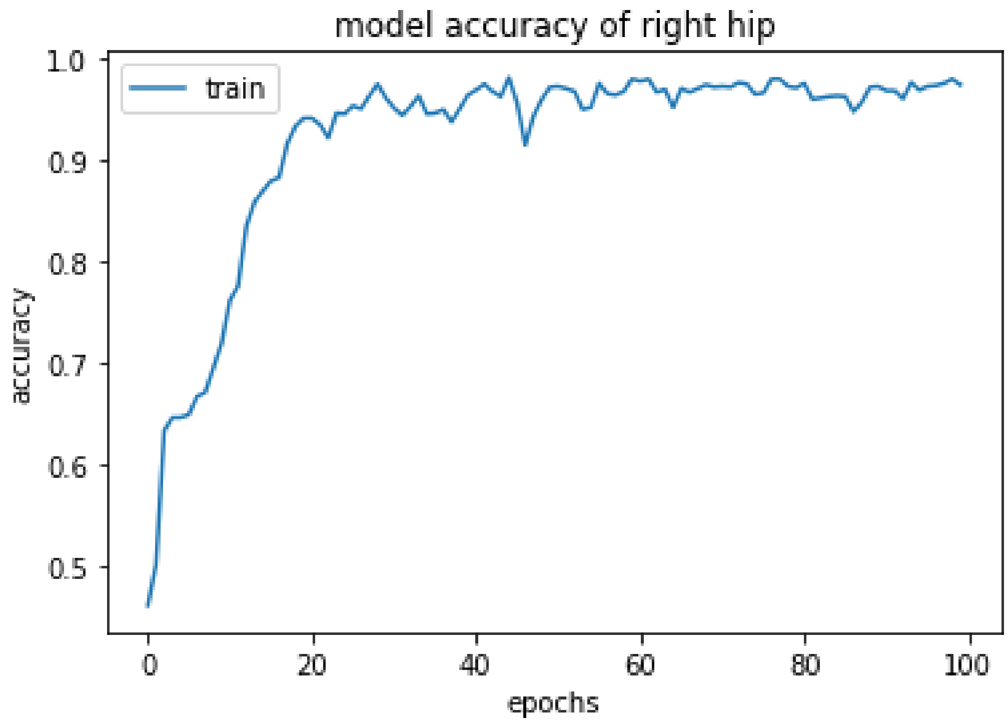


Fig 20

Neck

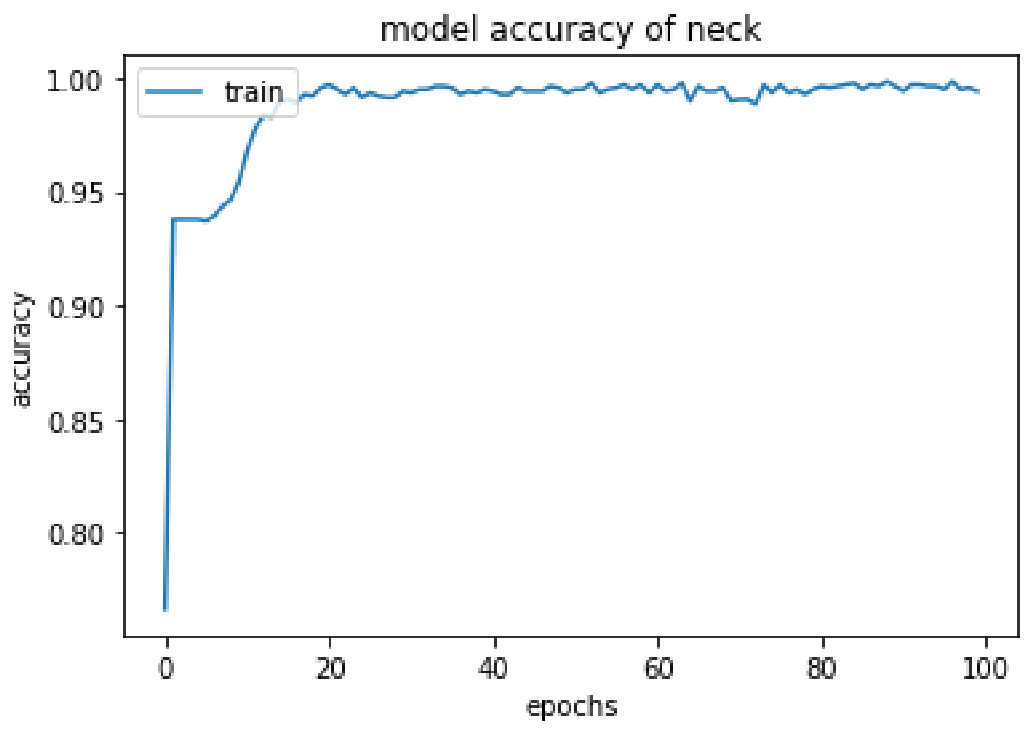


Fig 21

Head

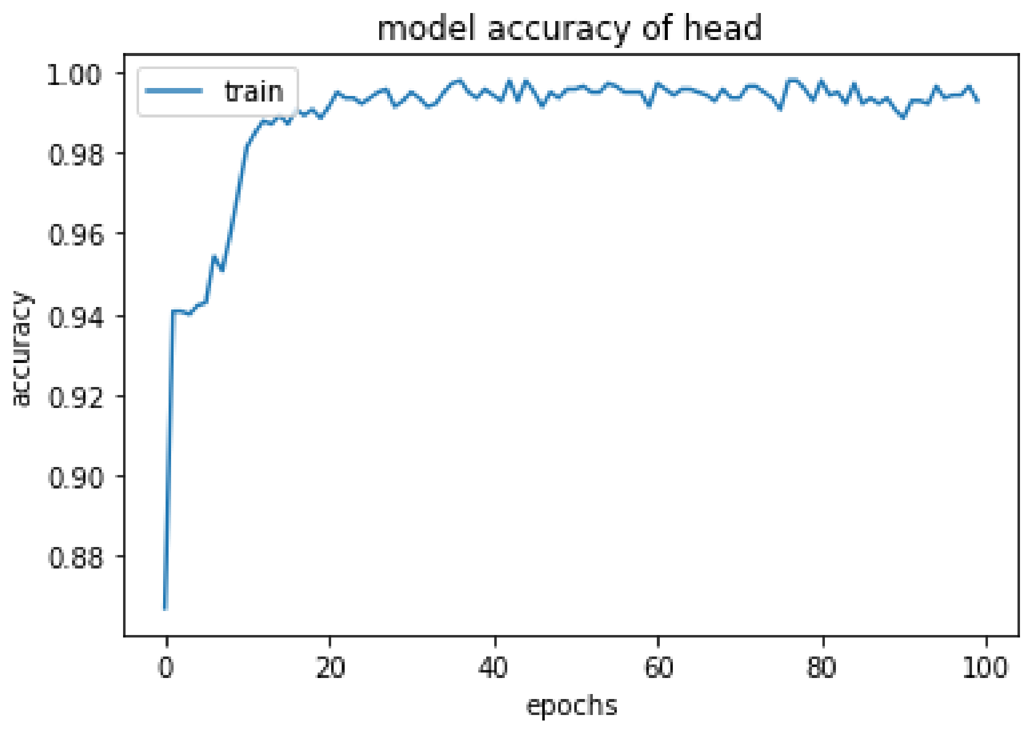


Fig 22

4.3 Loss

Left hip

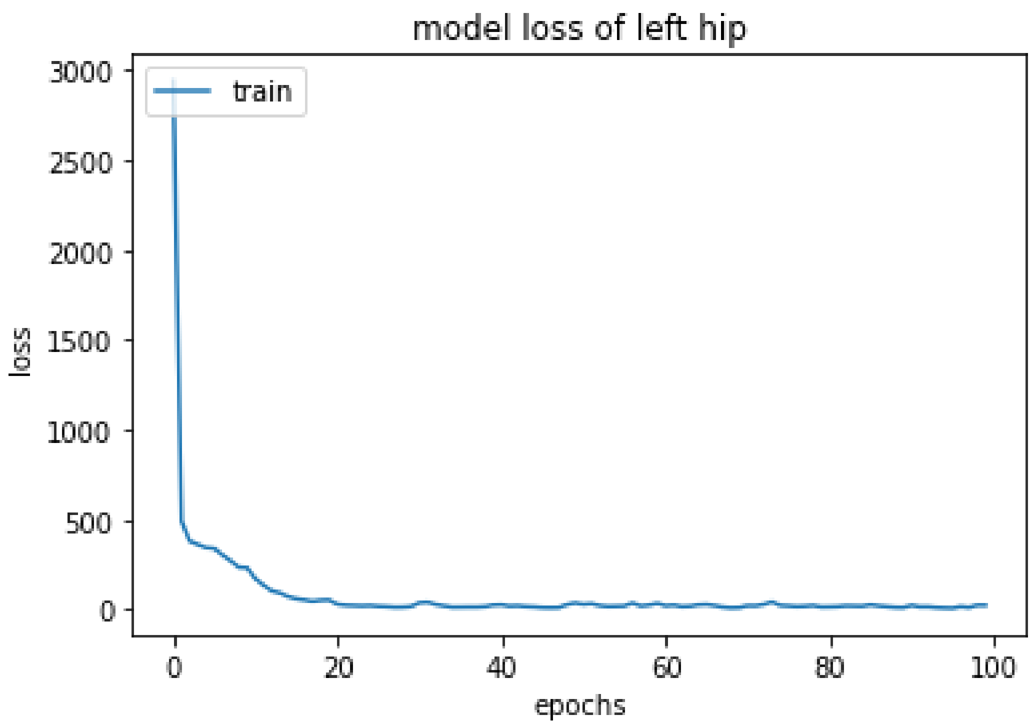


Fig 23

Left Knee

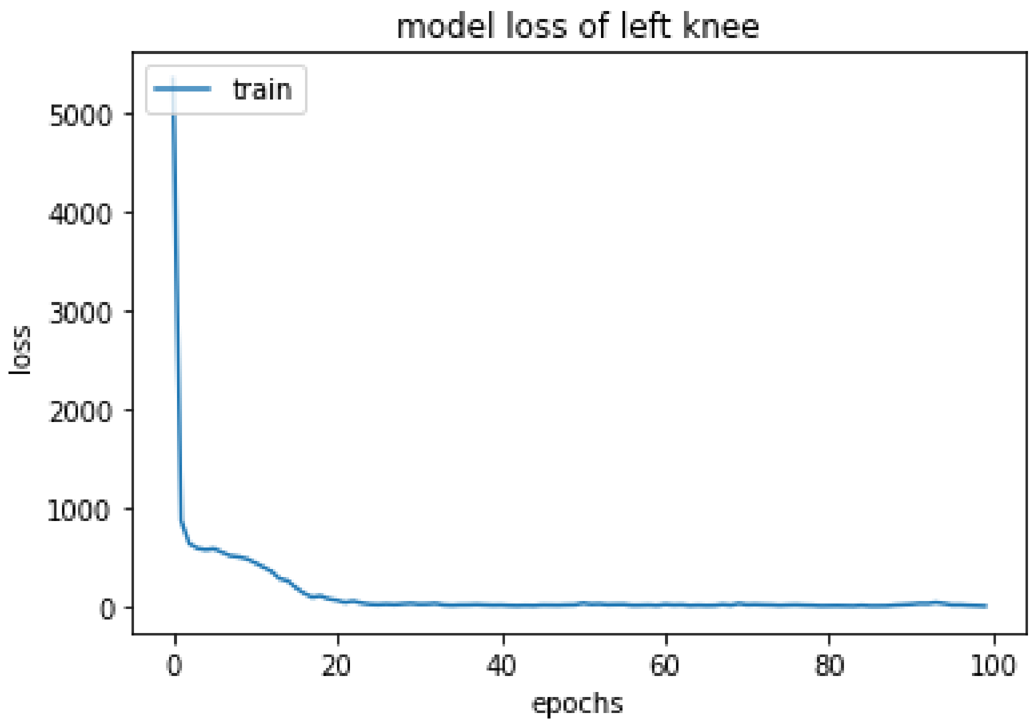


Fig 24

Left ankle

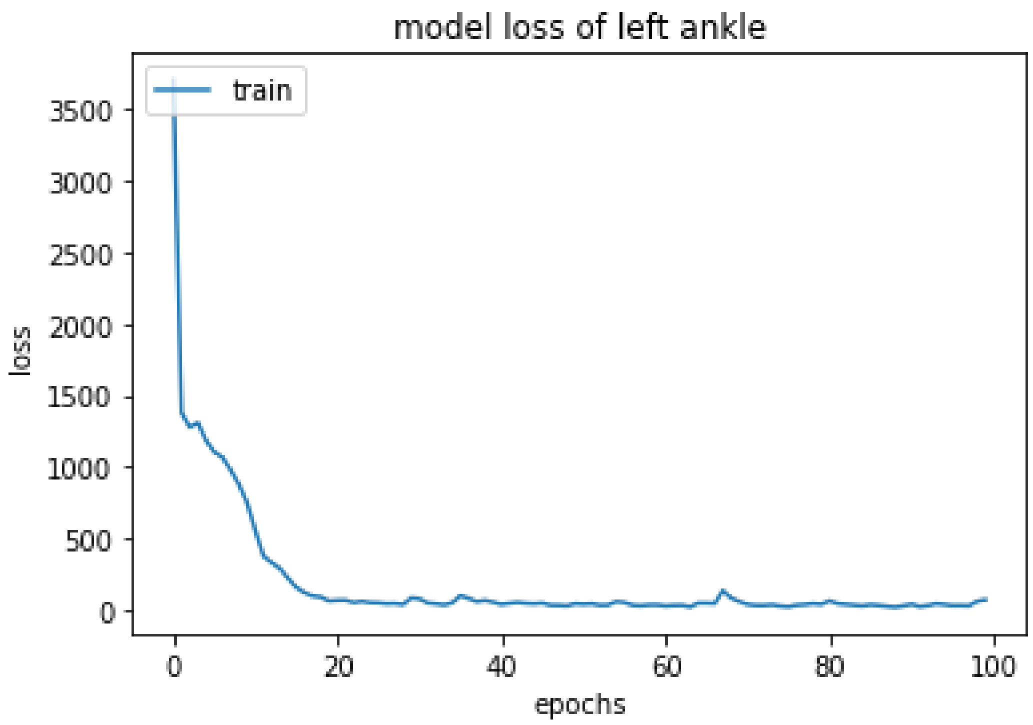


Fig 25

Left shoulder

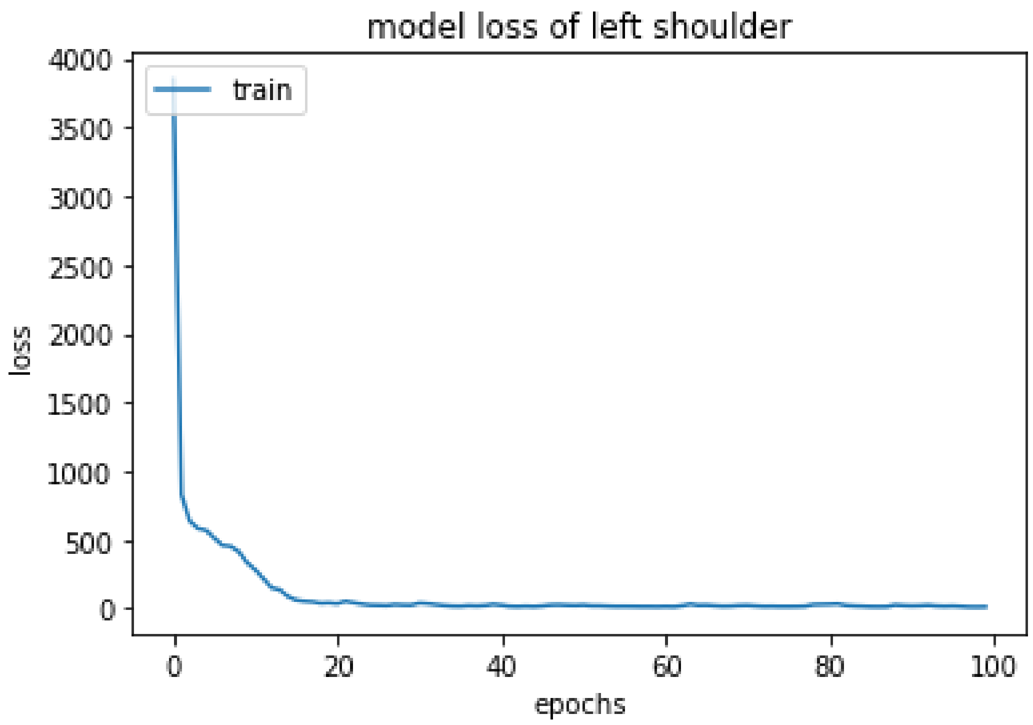


Fig 26

Left elbow

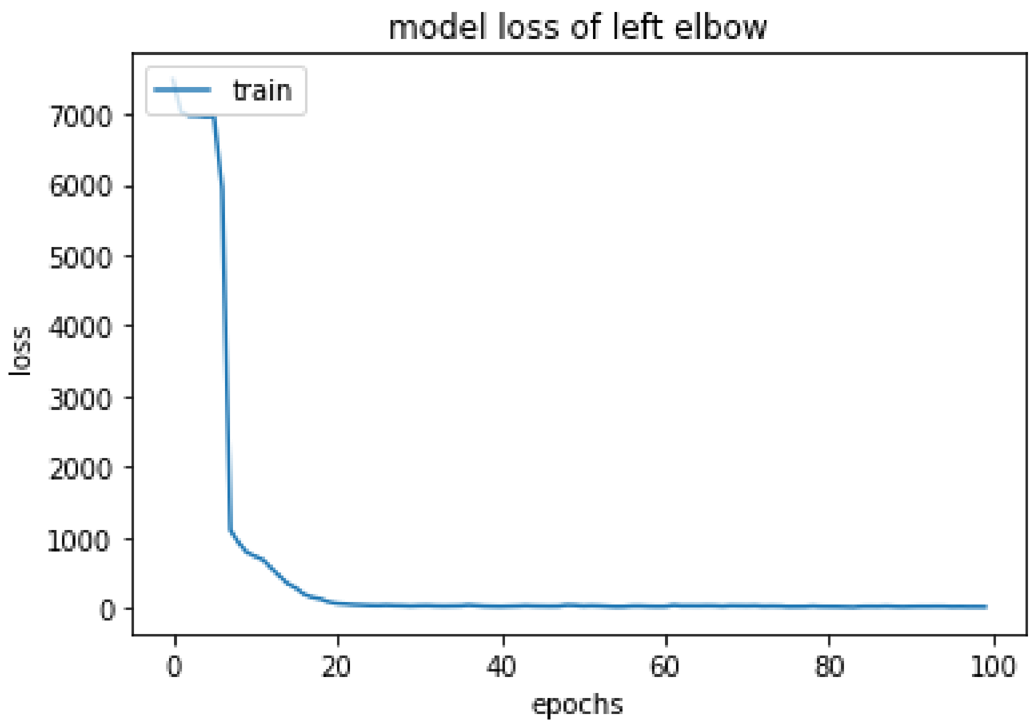


Fig 27

Left wrist

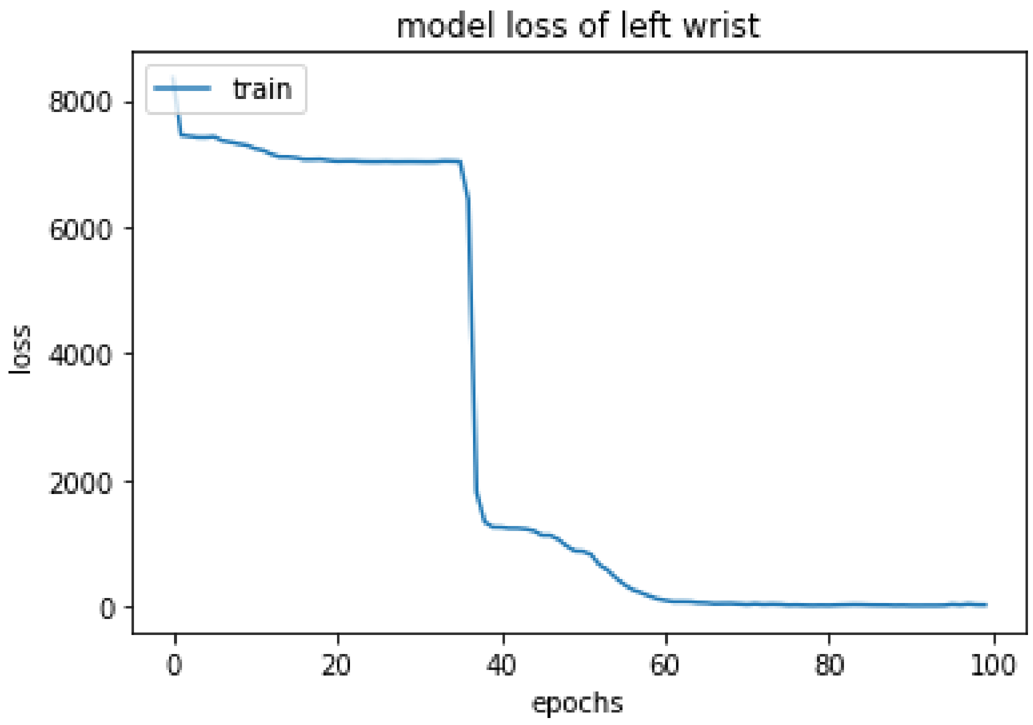


Fig 28

Right Wrist

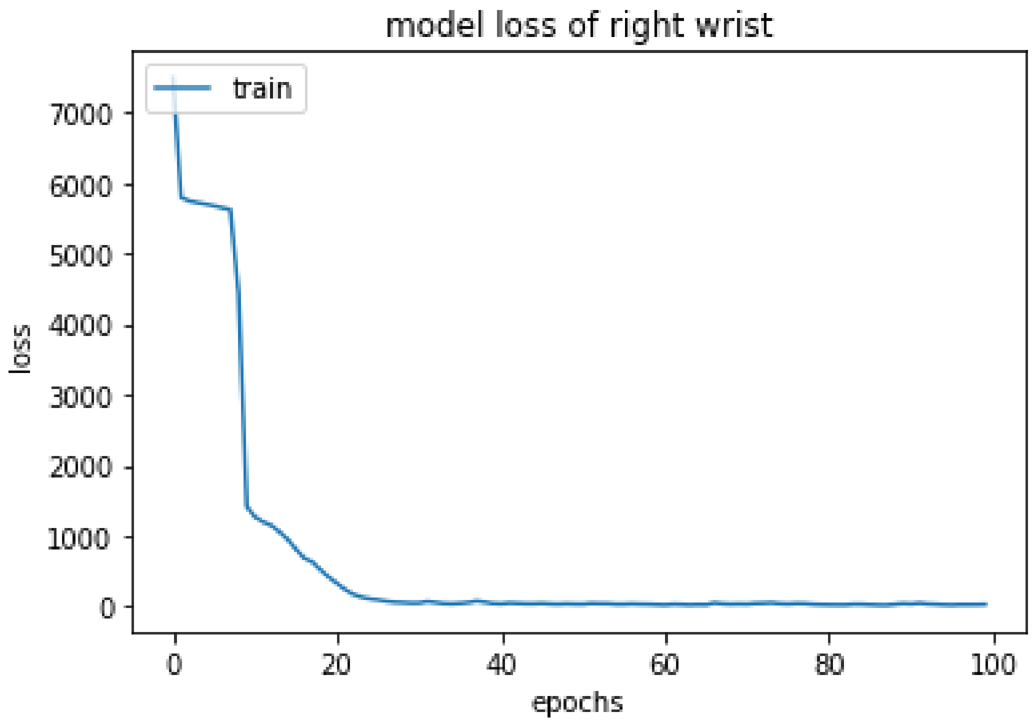


Fig 29

Right elbow

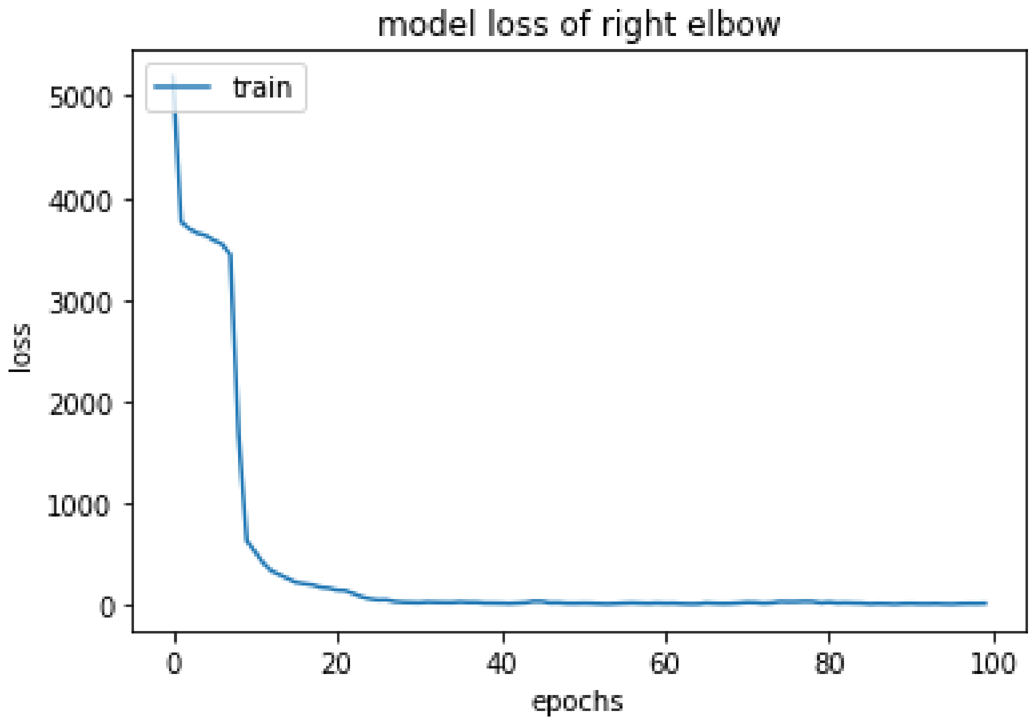


Fig 30

Right shoulder

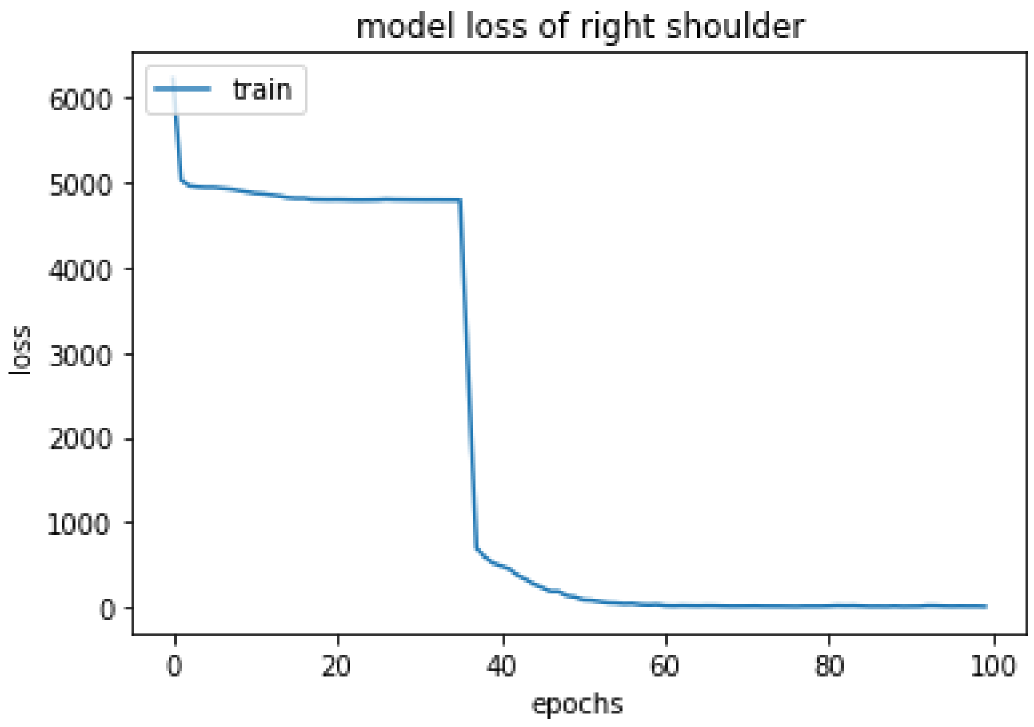


Fig 31

Right Knee

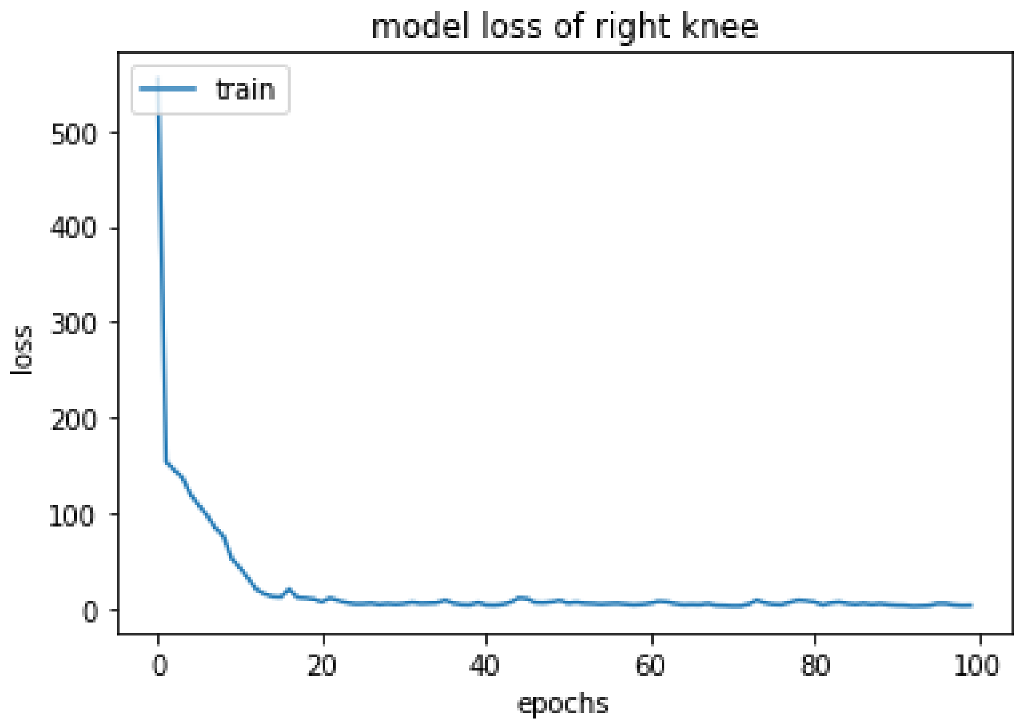


Fig 32

Right ankle

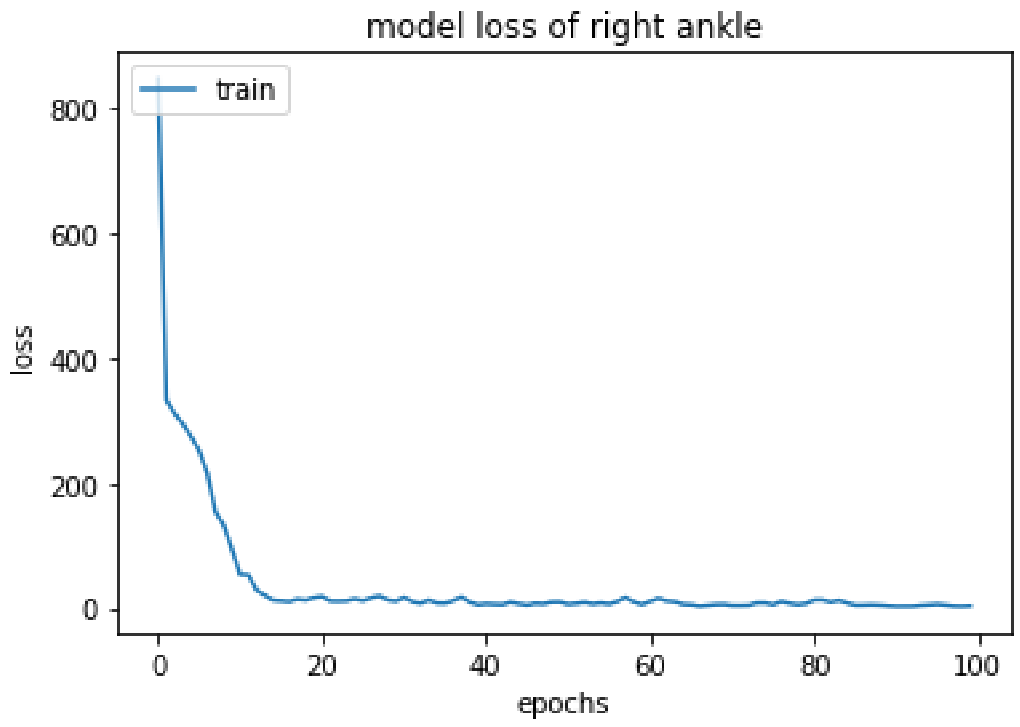


Fig 33

Right hip

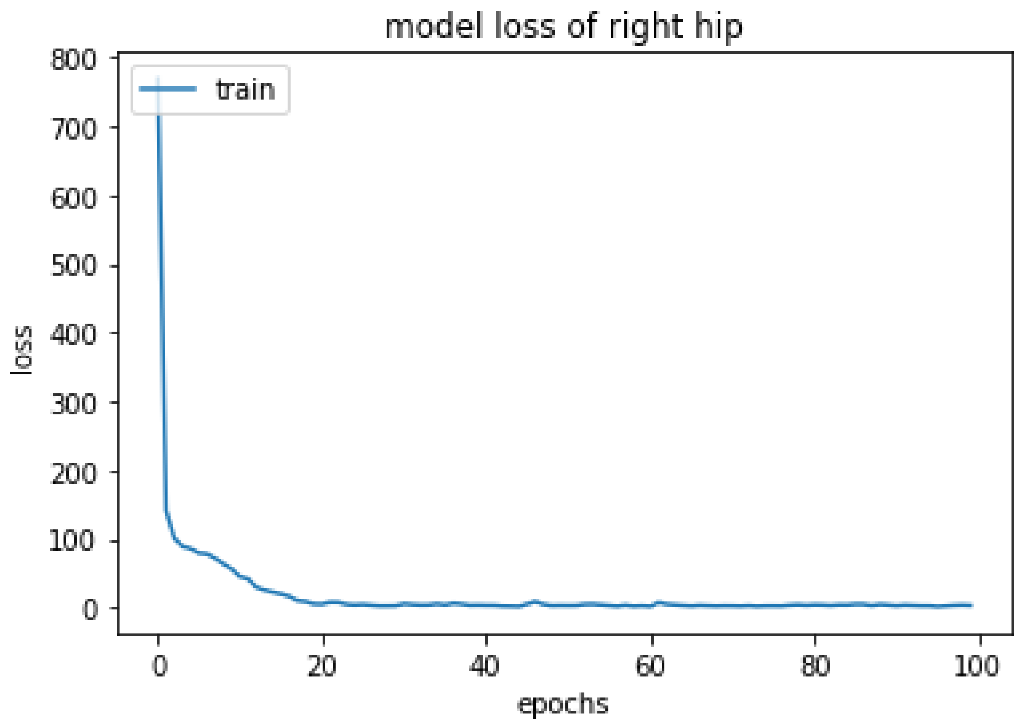


Fig 34

Neck

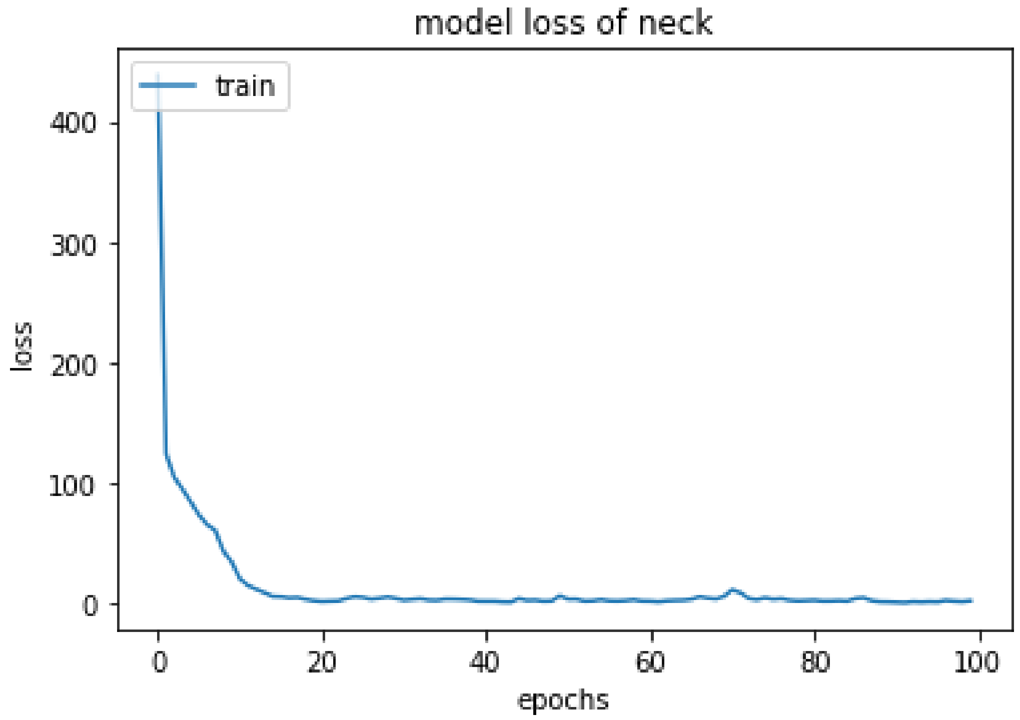


Fig 35

4.4 Comparison with other CNN architectures

4.4.1 Accuracy of the DeepPose model

<u>**Final Training and Testing accuracies**</u>

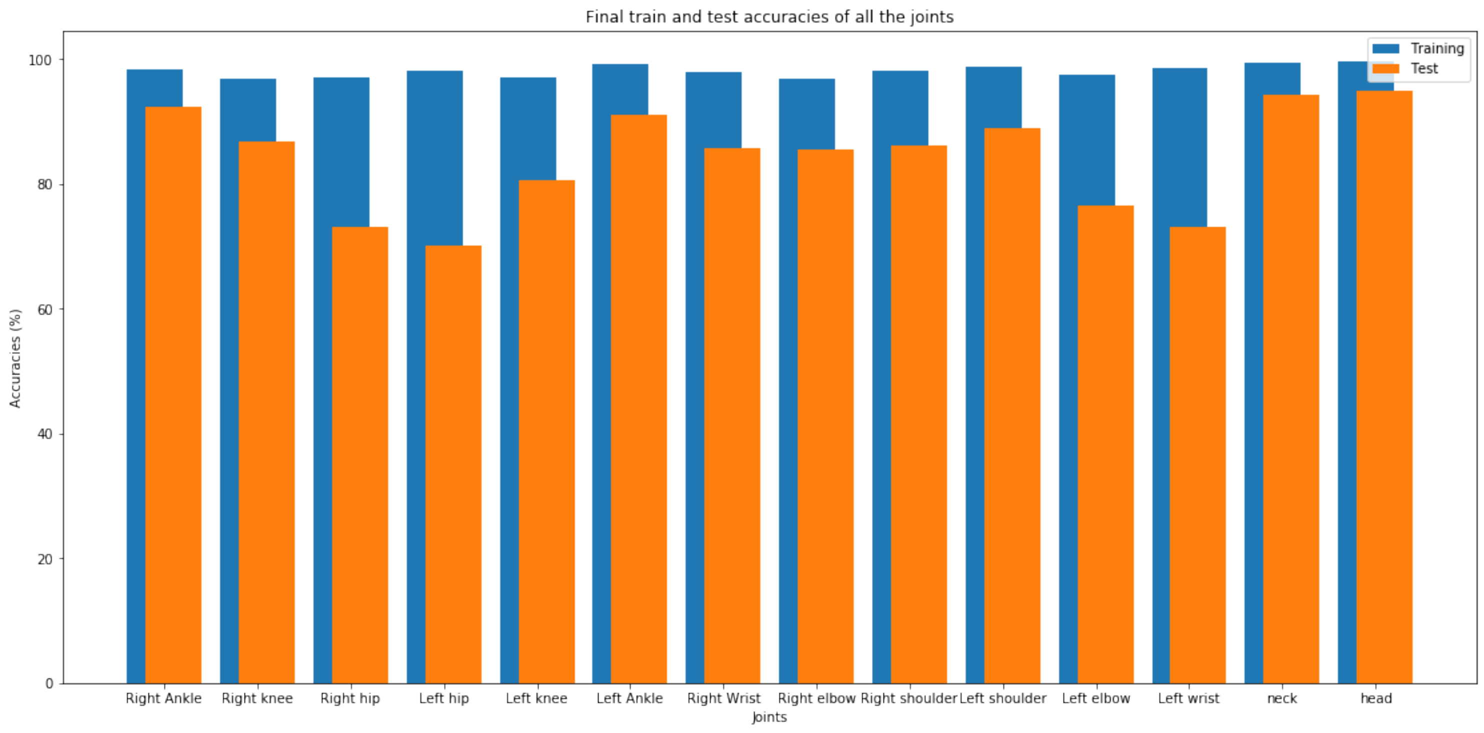


Fig 36

4.4.2 Accuracy of the ResNet34 model

<u>**Final Training and Testing accuracies**</u>

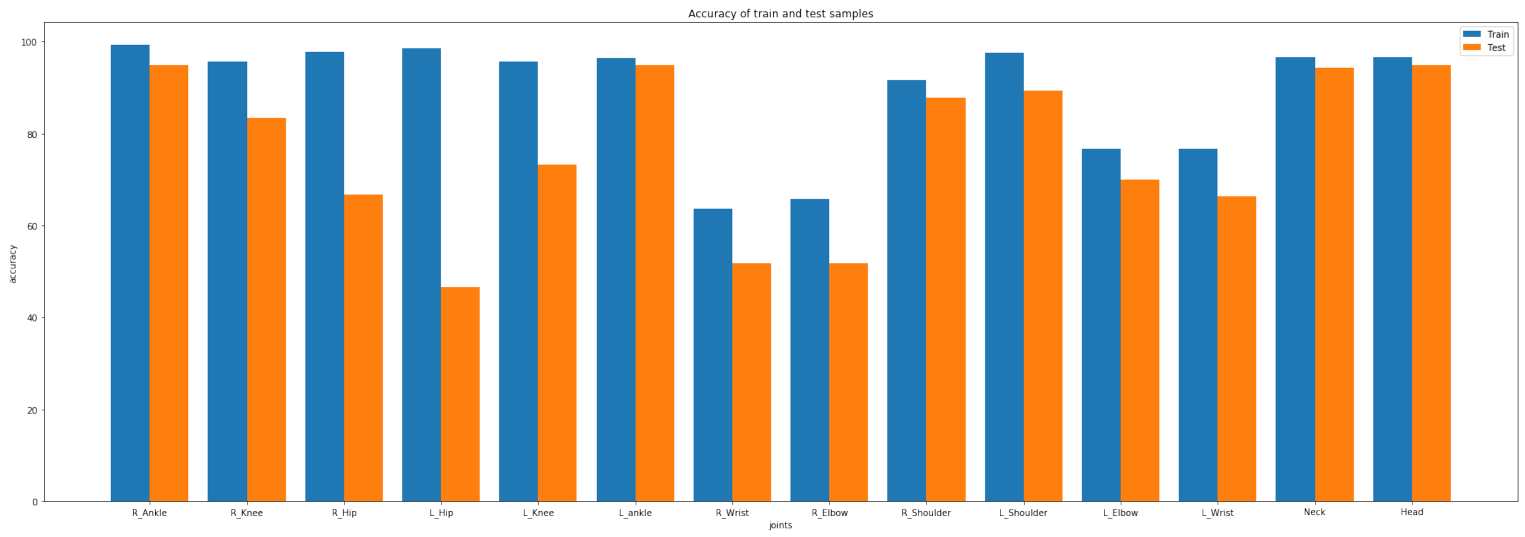


Fig 37

4.4.3 Accuracy of the AlexNet model

Final Training and Testing accuracies

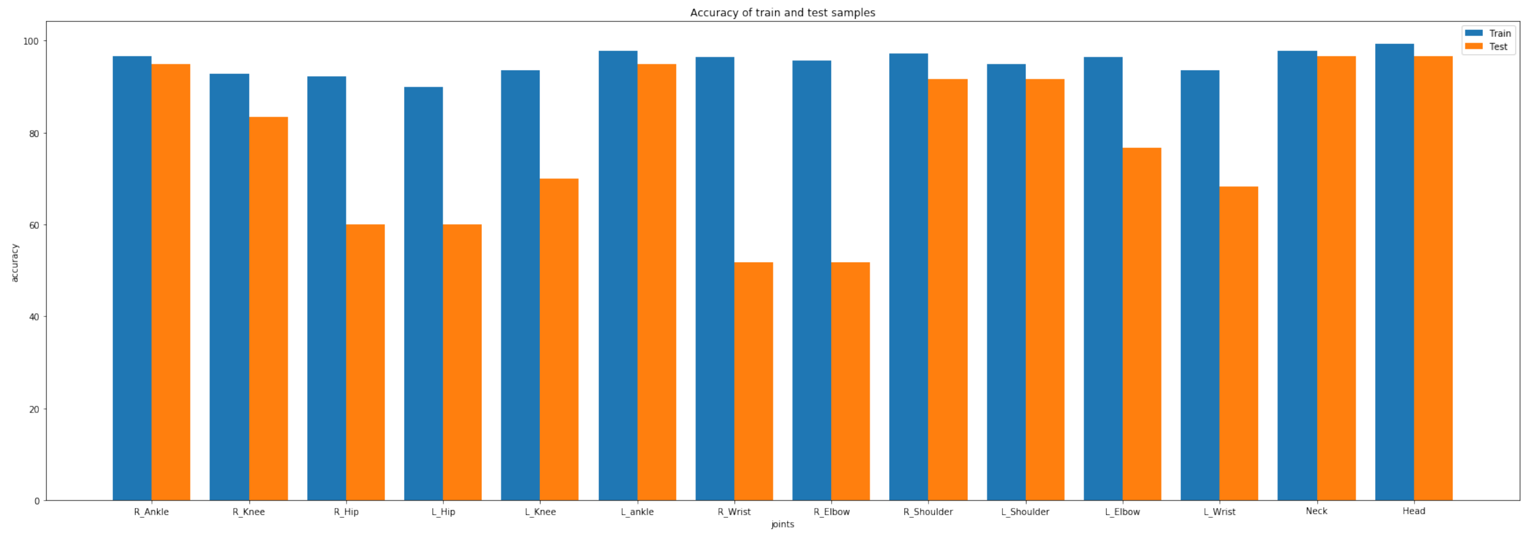


Fig 38

Chapter 5

Future Objectives

Following are the future objectives for our project.

5.1 Build and real time motion capture application

After we achieve enough accuracy on the model we plan to build a application that will be able to record real time motion capture data from a simple smartphone camera and this data can be used with many ways like transferring that data to a rigged character to make that virtual character to mimic the movements by animating that character according to the data. It can also be used to create full body AR filters and Controlling cybernetic/robotic arms with the captured data.

Chapter 6

Conclusion

This project helped us understand how pose estimation works and how to implement it using deep learning methods. We learnt how to implement it using the different approaches mentioned in the earlier research papers on this topic.Through these implementations, we were also able to come to the conclusion on which method performs the best. Through our own interpretation of these models, we were also able to change certain parameters to improve the performance of the model. We were also able to compare our own model with other existing CNN models, namely, ResNet and AlexNet. We came to the conclusion that different models perform differently based on the joint involved. For better training data, we also created a data annotation tool to get the same.

References

[1] Tomas Pfiste, James Charles, Andrew Zisserman,*Flowing ConvNets for Human Pose Estimation in Videos* , 2015.

[2] Alex Krizhevsky ,Ilya Sutskever,Geoffrey E. Hinton , *ImageNet Classification with Deep Convolutional Neural Networks*.

[3] Alexander Toshev,Christian Szegedy, *DeepPose: Human Pose Estimation via Deep Neural Networks*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun*Deep Residual Learning for Image Recognition*