Maulana Azad National Institute of Technology

Bhopal, India, 462003

Department of Computer Science and Engineering

MINOR PROJECT REPORT

Semester 6

Gait Analysis and pattern identification by calculating the hip, knee, ankle angles of the person per 100 steps at a given angular speed

Submitted by

Nikita Chauhan	**181112231**
Duppati Sahithi	**181112236**
Niharika Shilpi	**181112255**
Rajeshwari Pandravisam	**181112261**

Under the guidance of
Dr. Vaibhav Soni Sir

Department of Computer Science and Engineering

Session : 2020 - 2021

Maulana Azad National Institute of Technology

Bhopal, India, 462003

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project report carried out on **"Gait Analysis and pattern identification by calculating the hip, knee, ankle angles of the person per 100 steps at a given angular speed"** by the 3rd year students:

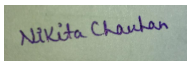
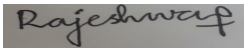
Nikita Chauhan	**181112231**
Duppati Sahithi	**181112236**
Niharika Shilpi	**181112255**
Rajeshwari Pandravisam	**181112261**

Have successfully completed their project in partial fulfilment of their Degree in Bachelor of Technology in Computer Science and Engineering.

Dr.Vaibhav Soni Sir
(Minor Project Mentor)

DECLARATION

We, hereby declare that the following report which is being presented in the Minor Project Documentation Entitled as "**Gait Analysis and pattern identification by calculating the hip, knee, ankle angles of the person per 100 steps at a given angular speed**" is an authentic documentation of our own original work and to the best of our knowledge. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization. Any contribution made to the research by others, with whom we have worked at Maulana Azad National Institute of Technology, Bhopal or elsewhere, is explicitly acknowledged in the report.

Nikita Chauhan	**181112231**	
Duppati Sahithi	**181112236**	
Niharika Shilpi	**181112255**	
Rajeshwari Pandravisam	**181112261**	

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide and coordinator **Dr.Vaibhav Soni Sir,** for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully.

It is imperative for us to mention the fact that the report of the minor project could not have been accomplished without the periodic suggestions and advice of our project guide **Dr.Vaibhav Soni Sir** and project coordinators **Dr. Dhirendra Pratap Singh** and **Dr. Jaytrilok Choudhary**.

We are also grateful to our respected director **Dr. N. S. Raghuwanshi** for permitting us to utilize all the necessary facilities of the college.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

ABSTRACT

A bipedal walking robot is a kind of humanoid robot. It is supposed to mimic human behavior and designed to perform human specific tasks.

But currently, humanoid robots are not capable of walking like human beings. To perform the walking task, in the current work, human gait data of 6 different walking styles named brisk walk, normal walk,

A very slow walk, medium walk, jogging and fast walk is collected through our configured IMU sensor and mobile-based accelerometer device.

To capture the pattern for 6 different walking styles, data is extracted for hip, knee, ankle, shank, thigh and foot. A total 6 classes of walking activities are explored for clinical examination. The accelerometer is placed at the center of the human body of 15 male and 10 female subjects.

In the experimental setup, we have done exploratory analysis over the different gait capturing techniques, different gait features and different gait classification techniques.

For the classification purpose,three state of art techniques are used as artificial neural networks, extreme learning machine and deep neural network learning based CNN mode. The model classification accuracy is obtained as 87.4%, 88% and 92%, respectively. Here, the WISDM activity data set is also used for verification purposes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROBLEM STATEMENT

Gait Analysis and pattern identification by calculating the hip, knee, ankle angles of the person per 100 steps at a given angular speed *i.e.* Neural Network Based Pattern identification of different human joints for different human walking styles by calculating the hip, knee, ankle angles of the person per 100 steps at a given angular speed when the person's information like age, height, mass, gait speed, leg length and gender is given.

INTRODUCTION

What is Gait?

Gait generation is the formulation and selection of a sequence of coordinated leg and body motions that propel a legged robot along a desired path. Approaches to gait generation can be classified into control, behavioral, rule-based, and constraint based paradigms.

What is Gait Analysis?

Gait analysis (GA) has been widely used to better understand the gait patterns of a wide range of populations. The application of this method has the ability to distinguish between normal and abnormal gaits , to determine the best intervention, and to detect pathologies at subclinical stages . These measures are objective and are typically performed using a three-dimensional (3D) motion-capture system and force plates.

How do legged mechanism and wheeled mechanism differ?

In comparison to wheeled mechanisms, legged mechanisms require complex design, move slowly, and are difficult to control. However, for locomotion over rough or discontinuous terrain, legged mechanisms are potentially superior to wheeled mechanisms.

Legged mechanisms make discrete terrain contacts and avoid undesirable footholds while wheeled mechanisms have rollers in continuous contact with the ground. The posture of a wheeled mechanism is dependent upon the terrain, but a legged mechanism can isolate its body from the terrain— achieving a more stable stance1 and allowing smooth level motion. Legged locomotion is theoretically more energy efficient because body propulsion does not expend as much energy in soil compaction and body motion can occur in a level plane decoupled from the effects of gravity

What are bipedal robots?

A bipedal walking robot is a type of humanoid robot which mimics like a human being and can be programmed to perform some tasks as required.This bipedal robot can assist humans to carry out the tasks or activities in hazardous environments.

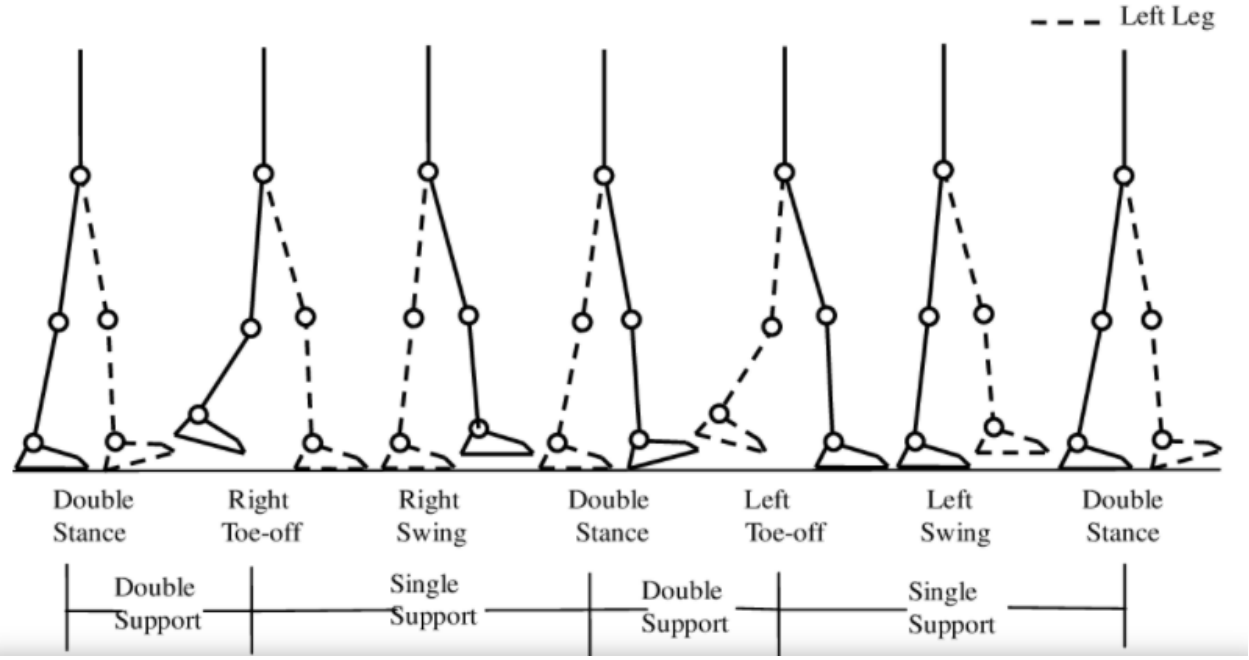


Figure 1

About the DATASET

Participants

Study participants included 42 volunteers, including 24 young adults (age 27.6 ± 4.4 years, height 171.1 ± 10.5 cm, and mass 68.4 ± 12.2 kg) and 18 older adults (age 62.7 ± 8.0 years, height 161.8 ± 9.5 cm, and mass 66.9 ± 10.1 kg). All participants were free of any lower-extremity injury in the last six months before the data were collected, and all were free of any orthopedic or neurologic disease that could interfere with their gait patterns.

Data Acquisition

Standard gait-analysis procedures were employed to collect data using a motion-capture system that had 12 cameras. The data were collected using a three-dimensional (3D) motion-capture system, force platforms and an instrumented treadmill while the subjects walked barefoot over a range of gait speeds.

Data Set

The data set comprises raw and processed (both in c3d and txt formats) pelvis and lower extremity kinematics and kinetics signals of 42 healthy volunteers (24 young adults and 18 older adults) walking in both overground and treadmill in a range of gait speeds. A file of metadata (xls), including anthropometrics and information related to each file in the data set is also provided. In addition, a model file (mdh) and a pipeline file (v3s) for the Visual 3D software program are also provided.

i)Raw Data

The files containing the raw data are available at Figshare (DOI: 10.6084/m9.figshare. 5722711) in both c3d format and ASCII file format. The total number of gait trials is not the same across participants because it reflects the variation in the number of valid trials per participant. The files provided are labeled “WBDS”, which stands for Walking Biomechanics Dataset; “xx”, for the participant’s assigned number (from 01 to 42); and “walk”, for the walking task. After this labeling, the following specific notations are used.

• Environment: “O” for overground and “T” for treadmill.

• Trial: “yy” indicates the trial number assignment for the overground condition only.

• Speed: “01” to “08”, which corresponds to the treadmill trials at 40%, 55%, 70%, 85%, 100%, 115%, 130%, and 145% of the self-selected, dimensionless speed (Froude number); and “S”, “C”, or “F”, which correspond to the slow, comfortable, and fast speeds for the overground trials.

ii) Meta Data

A metadata file named WBDSinfo.xlsx is available at Figshare and contains the following data in various columns (the bold word in each of the following items corresponds to the heading of a column).

1. Subject: the index of the subject (from 01 to 42).

2. FileName: the filename of the walking trial (WBDSxx, where xx identifies the participant), including the format extensions (*.c3d or *.txt).

3. AgeGroup: the “Young” or “Older” group.

4. Age: the participant’s age in years.

5. Height: the participant’s height in centimeters, measured with a calibrated stadiometer. 6. Mass: the participant’s body mass in kilograms, measured with a calibrated scale.

7. Gender: the participant’s gender (M or F).

8. Dominance: preferred leg for kicking a ball (R or L).

9. LegLength: leg length in centimeters (the average of the two legs).

10. Static1: whether the corresponding walking trial was assigned (Yes or No) to the Static1 file.

11. Static2: whether the corresponding walking trial was assigned (Yes or No) to the Static2 file. The Static2 was performed due to technical issues (e.g., markers dropping off during the session, markers needing to be repositioned, etc.).

12. GaitSpeed: the walking velocity at each trial (m/s).

13. TreadHands: whether the participant walked while hanging onto the treadmill handrails during the whole walking trial (Yes) or not at all (No).

14. FP_RightFoot: the force-plate number that the participant hit with the right foot.

15. FP_LeftFoot: the force-plate number that the participant hit with the left foot.

16. Notes: text strings with any notes about the treadmill or the overground trials (“–” if the trial has no notes). Ex: “FP3 signal presented offset”.

17. BorgScale: the corresponding Borg Scale value.

The dataset we used contains the walking style of 42 people of different age groups walking with 8 different angular speeds .

We observed how Pelvis, Hip , Knee, Ankle and Foot angles changed w.r.t x, y and z- axis and recorded the observations for 100 steps.

Input data contains the person's information like age , height, mass, gait speed, leg length, gender and expects the model to predict the corresponding walking style.

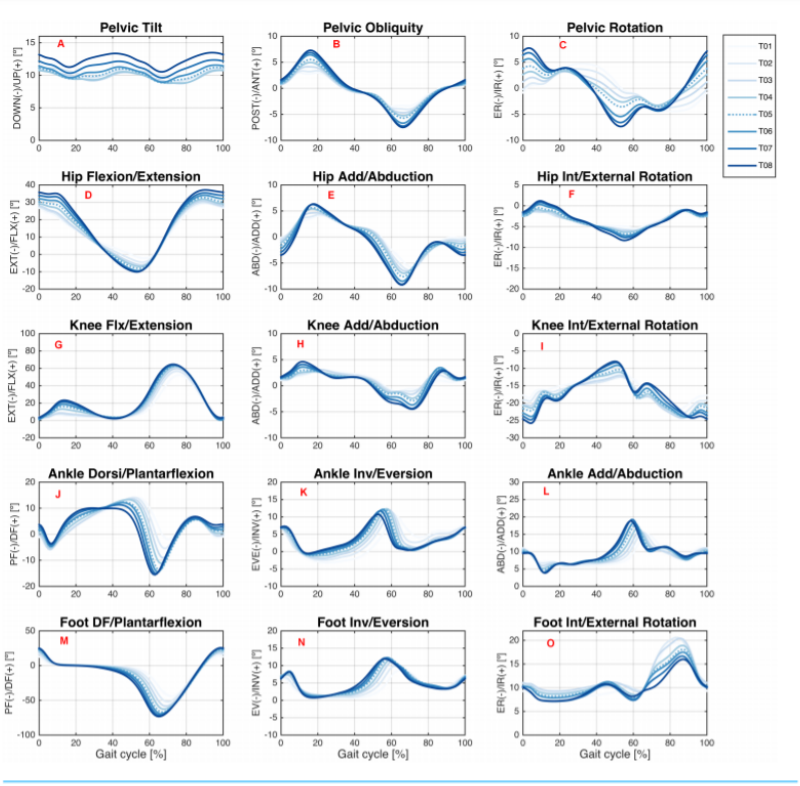


Figure 2 . Angular Kinematics during Treadmill walking

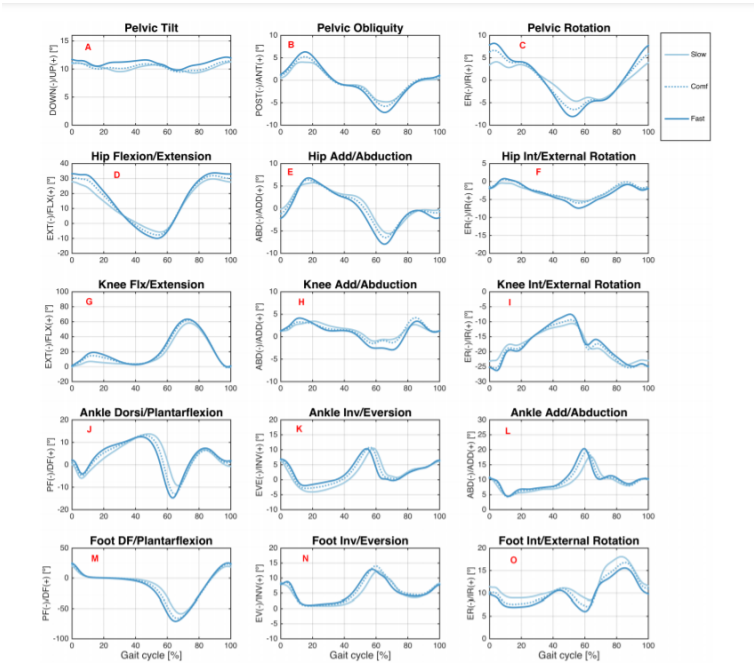


Figure 3 . Angular Kinematics during Overground walking

Combined Labeled Data Set Description:- The data set is collected for 25 subjects using wearable accelerometer (6-Degree IMU sensors) for 7 different activities named: Jogging, Normal walk, sit up, up stair walk, down stair walk, walk on heel and walk on toe for clinical examination purpose.

We have considered the subject from different age groups, gender and also collected data for pregnant women's data.

Each data file is having following column:

Time(s), ax(g), ay(g), az(g), wx(deg/s) , wy(deg/s), wz(deg/s), AngleX(deg) ,AngleY(deg) AngleZ(deg) T(°) .

Time(s) : time stamp

ax(g), ay(g), az(g): Acceleration about x,y and z axis

wx(deg/s) , wy(deg/s), wz(deg/s): Angular Velocity

AngleX(deg) ,AngleY(deg) AngleZ(deg)

T(°): Joint angle value in degree.

CODE

Importing Libraries

```
#Importing libraries which we'll be needing for the rest of our code
import sys
import numpy
import xlrd #To read excel file
import pandas as pd
import os
import matplotlib.pyplot as plt
import io
import math
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor
```

```
from scipy.signal import savgol_filter
from scipy import signal
from scipy.spatial import distance
from sklearn.model_selection import train_test_split

from tensorflow import keras, convert_to_tensor, float32, make_ndarray
from tensorflow.compat.v1 import Session
from tensorflow.keras import layers, backend
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Input, GlobalAveragePooling1D, Dropout
from tensorflow.keras.layers import LSTM, GRU, Conv1D, MaxPooling1D, MaxPool1D,
TimeDistributed, RepeatVector, Conv2D
from keras.layers.merge import concatenate
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
import random
```

Fetching the data

```
from google.colab import drive
drive.mount('/content/drive')
!cp /content/drive/MyDrive/TreadmilPublicData.zip .
! unzip TreadmilPublicData.zip
! cat TreadmilPublicData/WBDS42walkT08ang.txt
```

For plotting the data

```
def plotData(person, speed):
```

```
 ini_name = "TreadmilPublicData/"
 name ='WBDS' + ("" if person > 9 else "0") + str(person) + 'walkT' + "0" + str(speed)
 data = pd.read_csv(ini_name + name + 'ang.txt', delimiter = "\t")
 plt.plot(data['LKneeAngleZ'], label='Left Knee')
 plt.plot(data['LAnkleAngleZ'], label='Left Ankle')
 plt.plot(data['LHipAngleZ'], label='Left Hip')
 plt.ylabel("Angle")
 plt.xlabel("Time Steps")
 plt.legend()
   plt.show()
   print(data)


personalData = pd.read_excel(r"TreadmilPublicData/WBDSinfo.xlsx", engine='openpyxl')
filter = personalData["FileName"]=="WBDS01walkT01.c3d"


# filtering data
dt = personalData.loc[filter]


def todot(a,dt):
 x=len(a)
 b=numpy.zeros(a.shape)
 b[0]=a[0]/dt
 for i in range(1,x):
   b[i]=(a[i]-a[i-1])/dt
 return b
```

```
def plotDot(jointAngle, jointAngleDot, time):
  fig = plt.figure(figsize=(10,10))
  ax = fig.gca(projection='3d')
  z = numpy.linspace(0, time/100, time - 1)
  ax.plot(jointAngle, jointAngleDot, z)
  ax.set_xlabel("joint angle")
  ax.set_ylabel("angular velocity")
  plt.show()
```

Data Preprocessing

```
#input => [Age, Height, Mass, Gender, GaitSpeed, LegLength, Dominance, jointNumber]
#outputShape = 101 * 3 (hip, knee, ankle)
person1_x = []
person1_y = []
person2_x = []
person2_y = []
hip_test_x = []
ankle_test_x = []
knee_test_x = []
hip_test_y = []
ankle_test_y = []
knee_test_y = []
personal_information = []
def dataPrep():
  train_x, train_y, test_x, test_y = [], [], [], []
  prev = -1
  for i in range(1, 43):
```

```
  flagPerson = 0
  curr = random.randint(1, 8)
  for j in range(1, 9):
   temp_x = []
   # temp_y = []
   ini_name = "TreadmilPublicData/"
   name ='WBDS' + ("" if i > 9 else "0") + str(i) + 'walkT' + "0" + str(j)
   filter = personalData["FileName"]==name + ".c3d"
   dt = personalData.loc[filter]
   if flagPerson == 0:
    flagPerson = 1
    personal_information.append([dt.iloc[0]['Age'], dt.iloc[0]['Height'], dt.iloc[0]['Mass'],
                dt.iloc[0]['GaitSpeed(m/s)'], dt.iloc[0]['LegLength']])
   try:
    data = pd.read_csv(ini_name + name + 'ang.txt', delimiter = "\t")
    for joint in range(3):
     temp_x = []
     for _ in range(101):
      temp_x.append([dt.iloc[0]['Age'], dt.iloc[0]['Height'], dt.iloc[0]['Mass'], (0 if dt.iloc[0]['Gender']
== 'M' else 1),
                dt.iloc[0]['GaitSpeed(m/s)'], dt.iloc[0]['LegLength'], joint])


     if i == 1:
      person1_x.append(temp_x.copy())
     if i == 2:
      person2_x.append(temp_x.copy())


     if curr != j:
      # print("train =>", i, j)
```

```
      train_x.append(temp_x.copy())
      # train_y.append(temp_y.copy())
    else:
      test_x.append(temp_x.copy())
      if joint == 0:
        hip_test_x.append(temp_x.copy())
        hip_test_y.append(data['LHipAngleZ'])
      elif joint  == 1:
        knee_test_x.append(temp_x.copy())
        knee_test_y.append(data['LKneeAngleZ'])
      else:
        ankle_test_x.append(temp_x.copy())
        ankle_test_y.append(data['LAnkleAngleZ'])

  if curr != j:
    # train_y.append(temp_y.copy())
    train_y.append(data['LHipAngleZ'])
    train_y.append(data['LKneeAngleZ'])
    train_y.append(data['LAnkleAngleZ'])
  else:
    # test_y.append(temp_y.copy())
    test_y.append(data['LHipAngleZ'])
    test_y.append(data['LKneeAngleZ'])
    test_y.append(data['LAnkleAngleZ'])
  if i == 1:
    person1_y.append(data['LHipAngleZ'])
    person1_y.append(data['LKneeAngleZ'])
    person1_y.append(data['LAnkleAngleZ'])
  if i == 2:
```

```
        person2_y.append(data['LHipAngleZ'])
        person2_y.append(data['LKneeAngleZ'])
        person2_y.append(data['LAnkleAngleZ'])
    except:
      # print(str(e))
      continue

 train_x = numpy.array(train_x)
 train_y = numpy.array(train_y)
 test_x = numpy.array(test_x)
 test_y = numpy.array(test_y)
 return train_x, train_y, test_x, test_y



train_x, train_y, test_x, test_y = dataPrep()

n_outputs = 101
input_shape = train_x.shape[1:]

person1_x = numpy.array(person1_x)
person2_x = numpy.array(person2_x)
person1_y = numpy.array(person1_y)
person2_y = numpy.array(person2_y)

t = numpy.concatenate([train_x, test_x, person1_x, person2_x])
ty = numpy.concatenate([train_y, test_y, person1_y, person2_y])
print(t.shape, train_x.shape, test_x.shape)
```

```
t_max = t.max(axis=(0,1))
t_min = t.min(axis=(0,1))
ty_max = ty.max(axis=(0,1))
ty_min = ty.min(axis=(0,1))

# Normalizing
train_x = (train_x-t_min)/(t_max-t_min)
test_x = (test_x-t_min)/(t_max-t_min)
person1_x = (person1_x-t_min)/(t_max-t_min)
person2_x = (person2_x-t_min)/(t_max-t_min)
person1_y = (person1_y - ty_min)/(ty_max - ty_min)
person2_y = (person2_y - ty_min)/(ty_max - ty_min)
train_y = (train_y - ty_min)/(ty_max - ty_min)
test_y = (test_y - ty_min)/(ty_max - ty_min)
```

Working with different models

S.no.	Model we are going to make
1	**LSTM Model**
2	**CNN Model**
3	**GRU Model**
4	**Ensemble Model**
5	**VGG Model**

Table 1

1. LSTM Model

It is a special kind of recurrent neural network that is capable of learning long term dependencies in data. This is achieved because the recurring module of the **model** has a combination of four layers interacting with each other.

Creating the model

```
lstm_model = Sequential([
  LSTM(256, input_shape=input_shape),
  Dense(128, activation='relu'),
  Dropout(0.2),
  Dense(128, activation='relu'),
  Dropout(0.3),
  Dense(n_outputs, activation='relu')
])
```

Figure 4

Compiling the model

```
lstm_model.compile(optimizer='adam', loss='mse',metrics=['accuracy'])
lstm_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 256)               270336
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 101)               13029
=================================================================
Total params: 332,773
Trainable params: 332,773
Non-trainable params: 0
_________________________________________________________________
```

Figure 5

Fitting the model

```
[ ]
lstm_model_history = lstm_model.fit(train_x, train_y, epochs=200, validation_split=0.15, shuffle=True)
23/23 [==============================] - 7s 293ms/step - loss: 0.0043 - accuracy: 0.0602 - val_loss: 0.0041 - val_accuracy: 0.1231
Epoch 38/200
23/23 [==============================] - 7s 293ms/step - loss: 0.0041 - accuracy: 0.0919 - val_loss: 0.0044 - val_accuracy: 0.1615
Epoch 39/200
23/23 [==============================] - 7s 290ms/step - loss: 0.0044 - accuracy: 0.0977 - val_loss: 0.0042 - val_accuracy: 0.0923
Epoch 40/200
23/23 [==============================] - 7s 290ms/step - loss: 0.0043 - accuracy: 0.0753 - val_loss: 0.0050 - val_accuracy: 0.1000
Epoch 41/200
23/23 [==============================] - 7s 291ms/step - loss: 0.0041 - accuracy: 0.0971 - val_loss: 0.0046 - val_accuracy: 0.1000
Epoch 42/200
23/23 [==============================] - 7s 292ms/step - loss: 0.0040 - accuracy: 0.0804 - val_loss: 0.0036 - val_accuracy: 0.0769
Epoch 43/200
23/23 [==============================] - 7s 293ms/step - loss: 0.0039 - accuracy: 0.1026 - val_loss: 0.0047 - val_accuracy: 0.1000
Epoch 44/200
23/23 [==============================] - 7s 294ms/step - loss: 0.0040 - accuracy: 0.0947 - val_loss: 0.0045 - val_accuracy: 0.0923
Epoch 45/200
23/23 [==============================] - 7s 303ms/step - loss: 0.0041 - accuracy: 0.0817 - val_loss: 0.0047 - val_accuracy: 0.0615
```

Figure 6

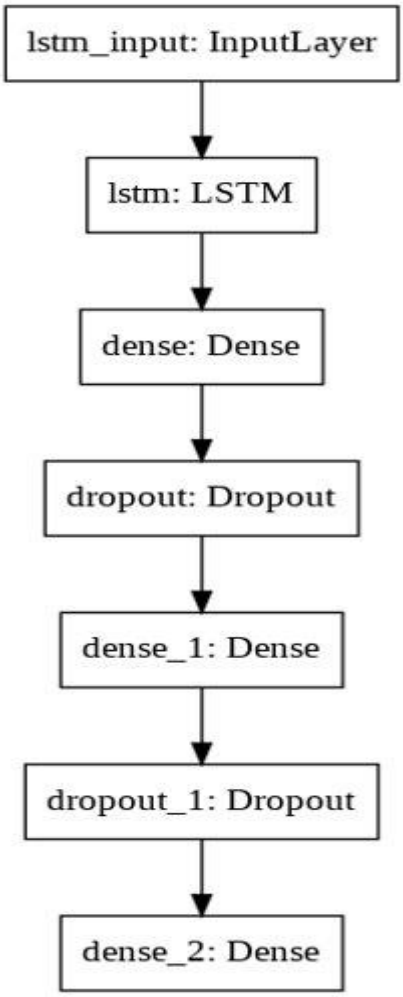

lstm_input: InputLayer
lstm: LSTM
dense: Dense
dropout: Dropout
dense_1: Dense
dropout_1: Dropout
dense_2: Dense

Figure 7

2. CNN Model

A Convolutional Neural Network (ConvNet/**CNN**) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

Creating and compiling the model

```
cnn_model = Sequential([
                        Conv1D(64,3, activation='relu', input_shape=input_shape),
                        MaxPooling1D(pool_size=2),
                        Flatten(),
                        Dense(128, activation='relu'),
                        Dropout(0.3),
                        Dense(n_outputs, activation='relu')
])

cnn_model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
cnn_model.summary()
```

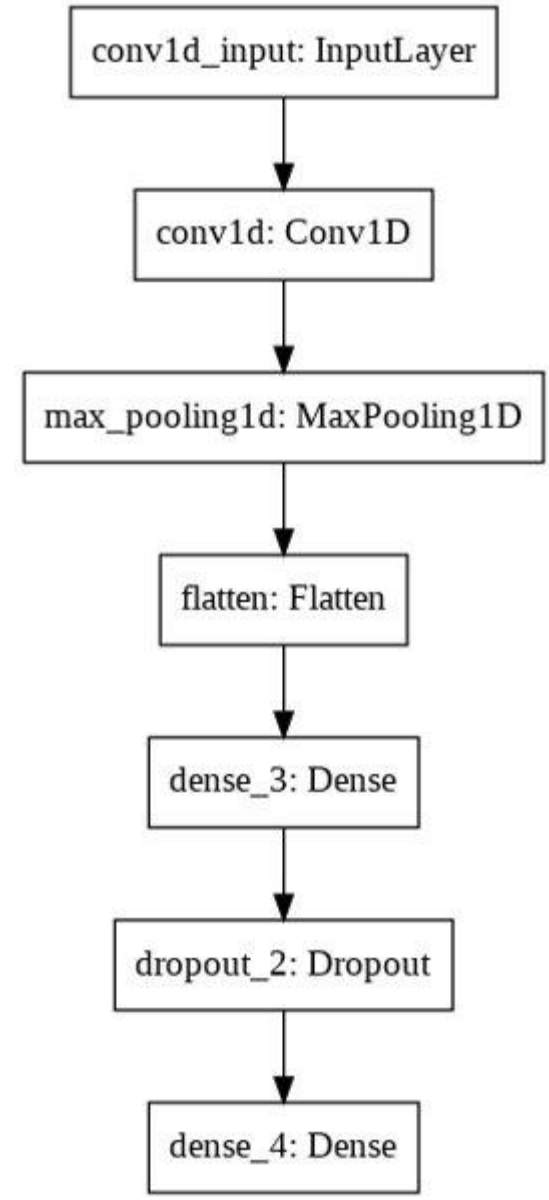
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 99, 64)            1408
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 49, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 101)               13029
=================================================================
Total params: 415,973
Trainable params: 415,973
Non-trainable params: 0
```


Figure 8

Fitting the model

```
cnn_model_history = cnn_model.fit(train_x, train_y, epochs=300, validation_split=0.15, shuffle=True)
Epoch 263/300
23/23 [==============================] - 0s 11ms/step - loss: 0.0020 - accuracy: 0.1620 - val_loss: 0.0067 - val_accuracy: 0.1154
Epoch 264/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0019 - accuracy: 0.1626 - val_loss: 0.0071 - val_accuracy: 0.1308
Epoch 265/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0021 - accuracy: 0.1699 - val_loss: 0.0072 - val_accuracy: 0.1308
Epoch 266/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0019 - accuracy: 0.1548 - val_loss: 0.0070 - val_accuracy: 0.1000
Epoch 267/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0020 - accuracy: 0.1847 - val_loss: 0.0064 - val_accuracy: 0.1308
Epoch 268/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0020 - accuracy: 0.1551 - val_loss: 0.0064 - val_accuracy: 0.1385
Epoch 269/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0023 - accuracy: 0.1761 - val_loss: 0.0061 - val_accuracy: 0.1615
Epoch 270/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0022 - accuracy: 0.1629 - val_loss: 0.0062 - val_accuracy: 0.1538
Epoch 271/300
23/23 [==============================] - 0s 12ms/step - loss: 0.0020 - accuracy: 0.1993 - val_loss: 0.0061 - val_accuracy: 0.1385
Epoch 272/300
23/23 [==============================] - 0s 13ms/step - loss: 0.0019 - accuracy: 0.1933 - val_loss: 0.0074 - val_accuracy: 0.1385
Epoch 273/300
```

Figure 9


conv1d_input: InputLayer
conv1d: Conv1D
max_pooling1d: MaxPooling1D
flatten: Flatten
dense_3: Dense
dropout_2: Dropout
dense_4: Dense

-

Figure 10

3. GRU Model

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural netwaorks.To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, update gate and reset gate. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or removing information which is irrelevant to the prediction.

Creating and compiling the model

```
2#CNN model
gru_model = Sequential([
                        GRU(128,  input_shape=input_shape),
                        Dense(128, activation='relu'),
                        Dropout(0.3),
                        Dense(n_outputs, activation='relu')
])

gru_model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
gru_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 128)               52608
_________________________________________________________________
dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 101)               13029
=================================================================
Total params: 82,149
Trainable params: 82,149
Non-trainable params: 0
_________________________________________________________________
```

Figure 11

Fitting the model

```
gru_model_history = gru_model.fit(train_x, train_y, epochs=300, validation_split=0.15, shuffle=True)
```

```
Epoch 271/300
23/23 [==============================] - 2s 86ms/step - loss: 0.0017 - accuracy: 0.1916 - val_loss: 0.0064 - val_accuracy: 0.1154
Epoch 272/300
23/23 [==============================] - 2s 84ms/step - loss: 0.0015 - accuracy: 0.1793 - val_loss: 0.0069 - val_accuracy: 0.1385
Epoch 273/300
23/23 [==============================] - 2s 86ms/step - loss: 0.0016 - accuracy: 0.1979 - val_loss: 0.0053 - val_accuracy: 0.1462
Epoch 274/300
23/23 [==============================] - 2s 89ms/step - loss: 0.0017 - accuracy: 0.1721 - val_loss: 0.0063 - val_accuracy: 0.1308
Epoch 275/300
23/23 [==============================] - 2s 90ms/step - loss: 0.0016 - accuracy: 0.2164 - val_loss: 0.0050 - val_accuracy: 0.1462
Epoch 276/300
23/23 [==============================] - 2s 90ms/step - loss: 0.0016 - accuracy: 0.1568 - val_loss: 0.0060 - val_accuracy: 0.1538
Epoch 277/300
23/23 [==============================] - 2s 91ms/step - loss: 0.0015 - accuracy: 0.1881 - val_loss: 0.0060 - val_accuracy: 0.1692
Epoch 278/300
23/23 [==============================] - 2s 89ms/step - loss: 0.0015 - accuracy: 0.2358 - val_loss: 0.0043 - val_accuracy: 0.1538
```

Figure 12

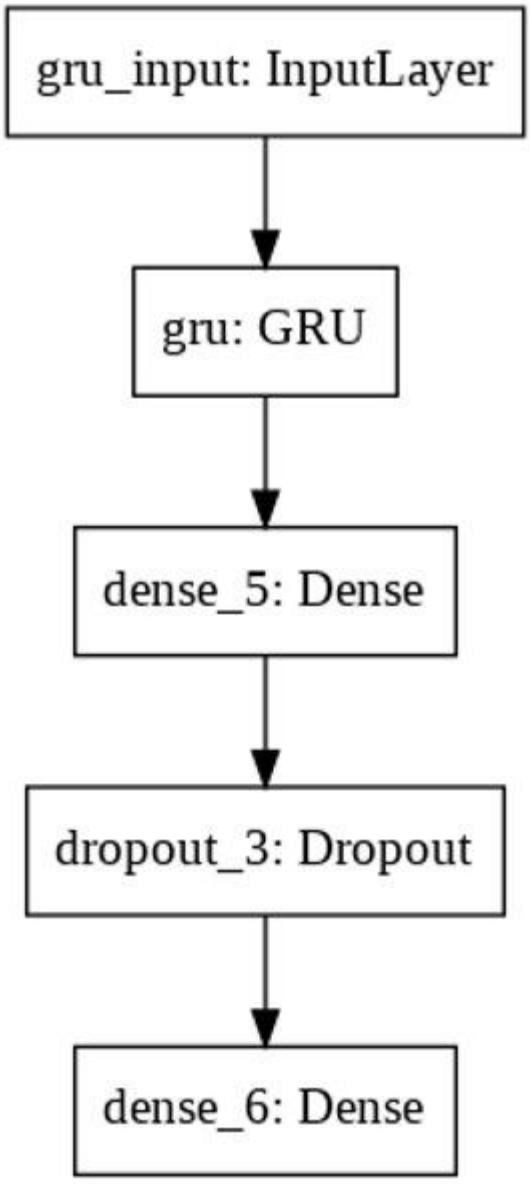

gru_input: InputLayer
gru: GRU
dense_5: Dense
dropout_3: Dropout
dense_6: Dense

Figure 13

4. Ensemble Model

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model.

So , we'll be using the above made three models combining their outcomes and predicting the most probable answer.

Creating the model

```
[ ]  cnn_model.trainable = False
     gru_model.trainable = False
     lstm_model.trainable = False
```

```
input = Input(shape=input_shape)
x1 = cnn_model(input)
x2 = gru_model(input)
x3 = lstm_model(input)
x = concatenate([x1,x2,x3])
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
output = Dense(n_outputs, activation='relu')(x)
```

```
model = Model(input, output)
```

Figure 14

Compiling the model

```
model.compile(optimizer='adam', loss='mse', metrics = ['accuracy'])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 101, 7)]     0
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 101)          415973      input_1[0][0]
__________________________________________________________________________________________________
sequential_2 (Sequential)       (None, 101)          82149       input_1[0][0]
__________________________________________________________________________________________________
sequential (Sequential)         (None, 101)          332773      input_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 303)          0           sequential_1[0][0]
                                                                 sequential_2[0][0]
                                                                 sequential[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 256)          77824       concatenate[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 256)          0           dense_7[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 101)          25957       dropout_4[0][0]
==================================================================================================
Total params: 934,676
Trainable params: 103,781
Non-trainable params: 830,895
__________________________________________________________________________________________________
```


Figure 15

Fitting the model

```
[ ] model_history = model.fit(train_x, train_y, epochs=300, validation_split=0.15, shuffle=True)
```

```
Epoch 56/300
23/23 [==============================] - 3s 127ms/step - loss: 0.0030 - accuracy: 0.0962 - val_loss: 0.0066 - val_accuracy: 0.1385
Epoch 57/300
23/23 [==============================] - 3s 126ms/step - loss: 0.0033 - accuracy: 0.1080 - val_loss: 0.0051 - val_accuracy: 0.1308
Epoch 58/300
23/23 [==============================] - 3s 125ms/step - loss: 0.0030 - accuracy: 0.0788 - val_loss: 0.0046 - val_accuracy: 0.1231
Epoch 59/300
23/23 [==============================] - 3s 124ms/step - loss: 0.0029 - accuracy: 0.0953 - val_loss: 0.0049 - val_accuracy: 0.1538
Epoch 60/300
23/23 [==============================] - 3s 125ms/step - loss: 0.0026 - accuracy: 0.1042 - val_loss: 0.0049 - val_accuracy: 0.1538
Epoch 61/300
23/23 [==============================] - 3s 124ms/step - loss: 0.0027 - accuracy: 0.0894 - val_loss: 0.0062 - val_accuracy: 0.1538
Epoch 62/300
23/23 [==============================] - 3s 124ms/step - loss: 0.0033 - accuracy: 0.1125 - val_loss: 0.0047 - val_accuracy: 0.1538
Epoch 63/300
23/23 [==============================] - 3s 126ms/step - loss: 0.0026 - accuracy: 0.1053 - val_loss: 0.0055 - val_accuracy: 0.1308
Epoch 64/300
23/23 [==============================] - 3s 125ms/step - loss: 0.0030 - accuracy: 0.1017 - val_loss: 0.0050 - val_accuracy: 0.1077
Epoch 65/300
23/23 [==============================] - 3s 125ms/step - loss: 0.0035 - accuracy: 0.1054 - val_loss: 0.0064 - val_accuracy: 0.1077
Epoch 66/300
```

Figure 16

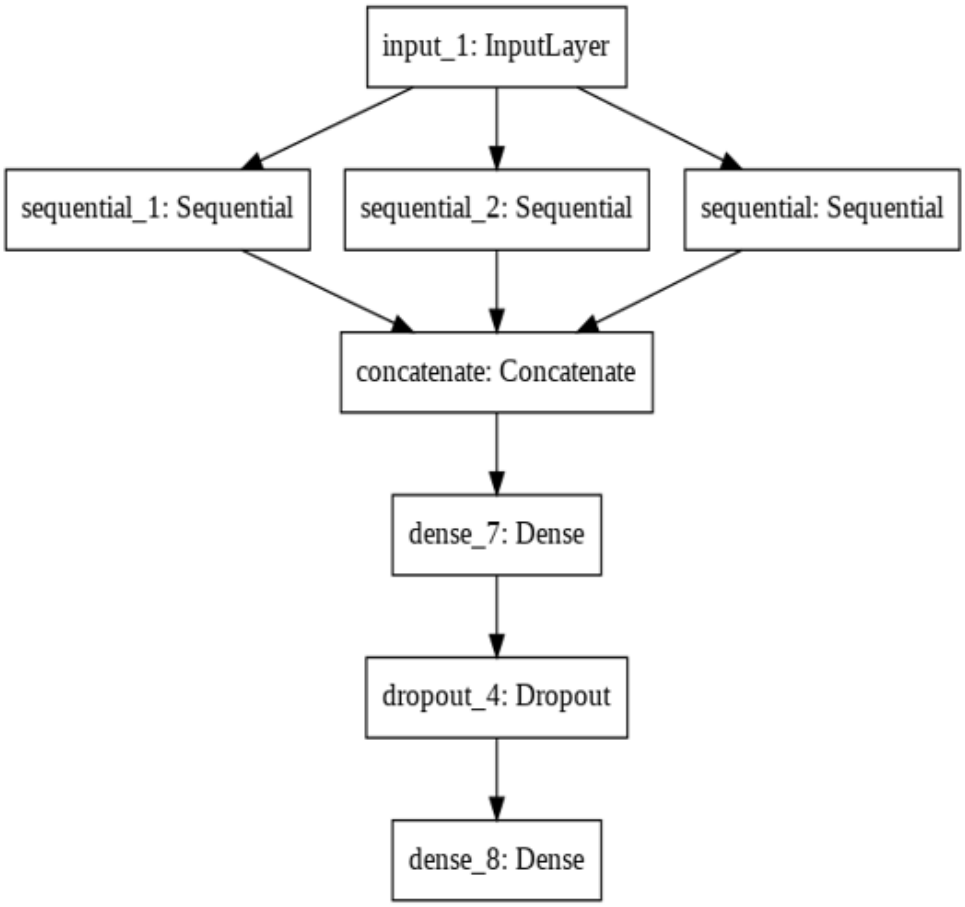


Figure 17

5. VGG Model

Creating and compiling the model

```
# function for creating a vgg block
def vgg_block(model,layer_in, n_filters, n_conv):
  # add convolutional layers
  for _ in range(n_conv):
    model.add(Conv1D(n_filters,3, padding='same', activation='relu'))
  # add max pooling layer
  model.add(MaxPooling1D(pool_size=2))
  return model


#create model
vgg_model=Sequential()
vgg_model.add(Conv1D(64,3,activation='relu',input_shape=input_shape))
# add vgg module
vgg_model= vgg_block(vgg_model,input_shape, 64, 3)
# add vgg module
vgg_model = vgg_block(vgg_model,(49, 64), 128, 3)
# add vgg module
vgg_model=vgg_block(vgg_model,(24,128), 256, 6)

vgg_model.add(Flatten())
vgg_model.add(Dense(128,activation='relu'))
vgg_model.add(Dropout(0.3))
vgg_model.add(Dense(n_outputs,activation='relu'))

vgg_model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
# summarize model
vgg_model.summary()
```


Figure 18

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_132 (Conv1D)          (None, 99, 64)            1408
_________________________________________________________________
conv1d_133 (Conv1D)          (None, 99, 64)            12352
_________________________________________________________________
conv1d_134 (Conv1D)          (None, 99, 64)            12352
_________________________________________________________________
conv1d_135 (Conv1D)          (None, 99, 64)            12352
_________________________________________________________________
max_pooling1d_38 (MaxPooling (None, 49, 64)            0
_________________________________________________________________
conv1d_136 (Conv1D)          (None, 49, 128)           24704
_________________________________________________________________
conv1d_137 (Conv1D)          (None, 49, 128)           49280
_________________________________________________________________
conv1d_138 (Conv1D)          (None, 49, 128)           49280
_________________________________________________________________
max_pooling1d_39 (MaxPooling (None, 24, 128)           0
_________________________________________________________________
conv1d_139 (Conv1D)          (None, 24, 256)           98560
_________________________________________________________________
conv1d_140 (Conv1D)          (None, 24, 256)           196864
_________________________________________________________________
conv1d_141 (Conv1D)          (None, 24, 256)           196864
_________________________________________________________________
conv1d_142 (Conv1D)          (None, 24, 256)           196864
_________________________________________________________________
conv1d_143 (Conv1D)          (None, 24, 256)           196864
_________________________________________________________________
conv1d_144 (Conv1D)          (None, 24, 256)           196864
_________________________________________________________________
max_pooling1d_40 (MaxPooling (None, 12, 256)           0
_________________________________________________________________
flatten_7 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               393344
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 101)               13029
=================================================================
Total params: 1,650,981
Trainable params: 1,650,981
Non-trainable params: 0
_________________________________________________________________
```

Figure 19

Fitting the model

```
vgg_history = vgg_model.fit(train_x, train_y, epochs=300, validation_split=0.15, shuffle=True)
```

```
Epoch 271/300
23/23 [==============================] - 6s 258ms/step - loss: 0.0021 - accuracy: 0.1464 - val_loss: 0.0064 - val_accuracy: 0.1462
Epoch 272/300
23/23 [==============================] - 6s 254ms/step - loss: 0.0022 - accuracy: 0.1477 - val_loss: 0.0060 - val_accuracy: 0.0846
Epoch 273/300
23/23 [==============================] - 6s 255ms/step - loss: 0.0021 - accuracy: 0.1286 - val_loss: 0.0057 - val_accuracy: 0.1154
Epoch 274/300
23/23 [==============================] - 6s 256ms/step - loss: 0.0021 - accuracy: 0.1477 - val_loss: 0.0060 - val_accuracy: 0.1077
Epoch 275/300
23/23 [==============================] - 6s 254ms/step - loss: 0.0020 - accuracy: 0.1409 - val_loss: 0.0052 - val_accuracy: 0.1308
Epoch 276/300
23/23 [==============================] - 6s 257ms/step - loss: 0.0020 - accuracy: 0.1313 - val_loss: 0.0052 - val_accuracy: 0.1385
Epoch 277/300
23/23 [==============================] - 6s 254ms/step - loss: 0.0020 - accuracy: 0.1628 - val_loss: 0.0065 - val_accuracy: 0.1538
Epoch 278/300
23/23 [==============================] - 6s 255ms/step - loss: 0.0020 - accuracy: 0.1532 - val_loss: 0.0050 - val_accuracy: 0.1769
Epoch 279/300
23/23 [==============================] - 6s 254ms/step - loss: 0.0018 - accuracy: 0.1409 - val_loss: 0.0062 - val_accuracy: 0.1231
Epoch 280/300
23/23 [==============================] - 6s 257ms/step - loss: 0.0021 - accuracy: 0.1327 - val_loss: 0.0079 - val_accuracy: 0.1231
Epoch 281/300
23/23 [==============================] - 6s 255ms/step - loss: 0.0020 - accuracy: 0.1532 - val_loss: 0.0057 - val_accuracy: 0.1231
Epoch 282/300
23/23 [==============================] - 6s 256ms/step - loss: 0.0020 - accuracy: 0.1505 - val_loss: 0.0084 - val_accuracy: 0.1385
Epoch 283/300
23/23 [==============================] - 6s 258ms/step - loss: 0.0021 - accuracy: 0.1546 - val_loss: 0.0061 - val_accuracy: 0.0846
Epoch 284/300
23/23 [==============================] - 6s 263ms/step - loss: 0.0020 - accuracy: 0.1601 - val_loss: 0.0070 - val_accuracy: 0.1308
Epoch 285/300
23/23 [==============================] - 6s 260ms/step - loss: 0.0021 - accuracy: 0.1477 - val_loss: 0.0062 - val_accuracy: 0.0846
```


Figure 20

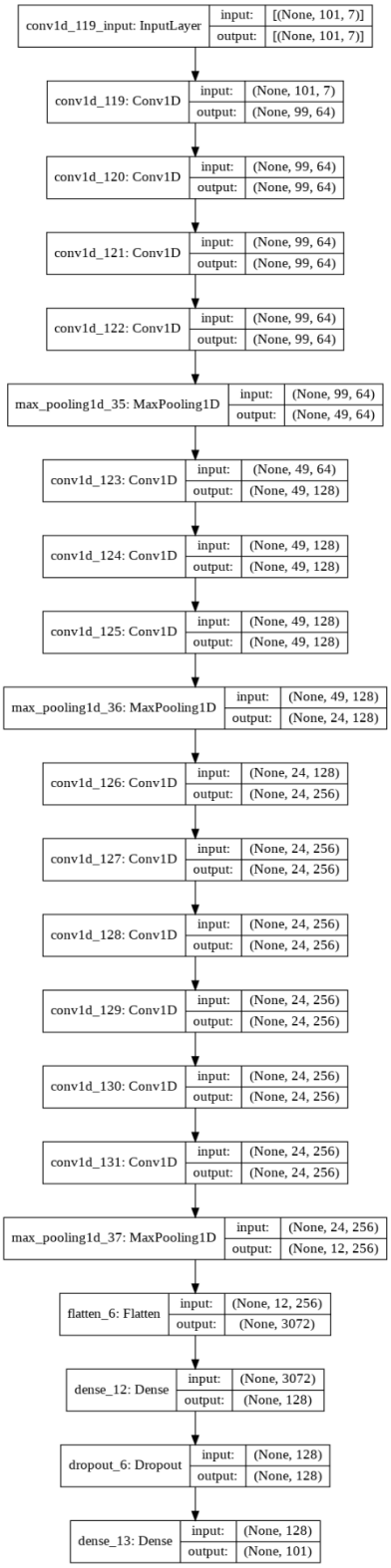

conv1d_119_input: InputLayer
input: [(None, 101, 7)]
output: [(None, 101, 7)]
conv1d_119: Conv1D
input: (None, 101, 7)
output: (None, 99, 64)
conv1d_120: Conv1D
input: (None, 99, 64)
output: (None, 99, 64)
conv1d_121: Conv1D
input: (None, 99, 64)
output: (None, 99, 64)
conv1d_122: Conv1D
input: (None, 99, 64)
output: (None, 99, 64)
max_pooling1d_35: MaxPooling1D
input: (None, 99, 64)
output: (None, 49, 64)
conv1d_123: Conv1D
input: (None, 49, 64)
output: (None, 49, 128)
conv1d_124: Conv1D
input: (None, 49, 128)
output: (None, 49, 128)
conv1d_125: Conv1D
input: (None, 49, 128)
output: (None, 49, 128)
max_pooling1d_36: MaxPooling1D
input: (None, 49, 128)
output: (None, 24, 128)
conv1d_126: Conv1D
input: (None, 24, 128)
output: (None, 24, 256)
conv1d_127: Conv1D
input: (None, 24, 256)
output: (None, 24, 256)
conv1d_128: Conv1D
input: (None, 24, 256)
output: (None, 24, 256)
conv1d_129: Conv1D
input: (None, 24, 256)
output: (None, 24, 256)
conv1d_130: Conv1D
input: (None, 24, 256)
output: (None, 24, 256)
conv1d_131: Conv1D
input: (None, 24, 256)
output: (None, 24, 256)
max_pooling1d_37: MaxPooling1D
input: (None, 24, 256)
output: (None, 12, 256)
flatten_6: Flatten
input: (None, 12, 256)
output: (None, 3072)
dense_12: Dense
input: (None, 3072)
output: (None, 128)
dropout_6: Dropout
input: (None, 128)
output: (None, 128)
dense_13: Dense
input: (None, 128)
output: (None, 101)

Figure 21

Comparison among all the models

Plot of how the ankle , knee and hip angles of the person changes for 100 steps by all the models and comparing tha with different values

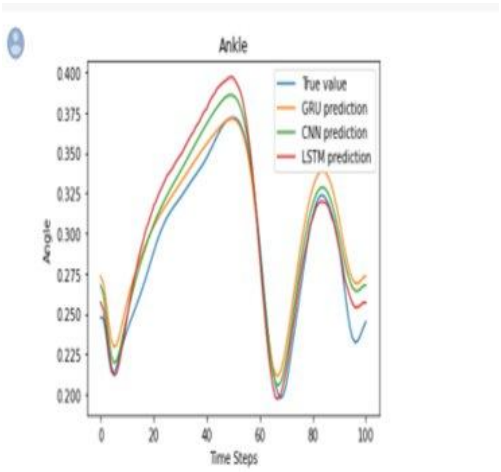


Figure 22

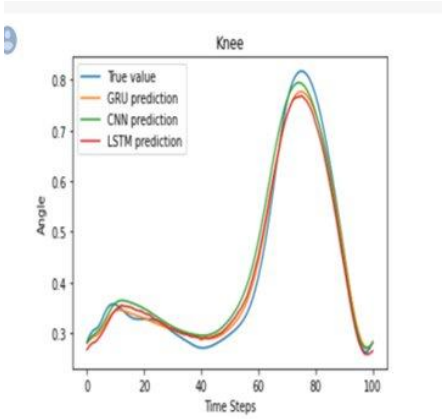


Figure 23

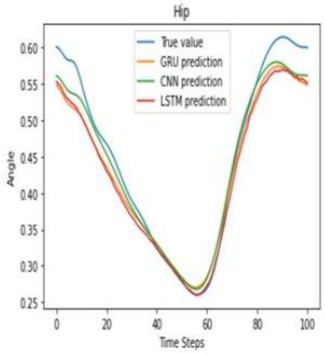


Figure 24

S.no.	Model Name	Hip_mean_error	Knee_mean_error	Ankle_mean_error
1	LSTM	0.2806581851159908	0.0351174787052302	0.024801056202634574
2	CNN	0.029615926688723773	0.04001808845529718	0.026002174149845
3	GRU	0.02411768486540539	0.03048257636745935	0.024127558921955096
4	Ensemble	0.03201705333297378	0.03801530859388143	0.02617658254016726
5	VGG	0.03834686734542258	0.03903141846883398	0.028249844659730457

Table 2

Conclusion:

We observe that the mean error of the GRU Model is comparatively less as compared with LSTM, GRU, CNN, Ensemble and VGG Model when calculated for knee, hip and ankle. Hence we conclude that GRU Model is performing well among the rest of the models.

Application:

Currently, humanoid robots are not capable of walking like human beings. But with our research we can make robots walk like humans just by changing the angle and position of the ankle, hip and knee of the humanoid robots.

Future Scope:

1. Currently the maximum accuracy of all models is 92%. We can use some other models and parameters to increase the accuracy of the model and so of our project.
2. The dataset we used in our project is of human walk on a treadmill and overground, for further improvement in bipedal robots we can take a dataset where we make our objects(human) walk on an inclined plane like stairs, mountain etc,.

References

1. **Semwal, V.B., Gaud, N., Lalwani, P. *et al***. Pattern identification of different human joints for different human walking styles using inertial measurement unit (IMU) sensors. *Artif Intell Rev* (2021). https://doi.org/10.1007/s10462-021-09979-x
2. **Fukuchi CA, Fukuchi RK, Duarte M. 2018.** A public dataset of overground and treadmill walking kinematics and kinetics in healthy individuals**.** *PeerJ* 6:e4640 https://doi.org/10.7717/peerj.4640
3. **Fukuchi, Claudiane; Fukuchi, Reginaldo; Duarte, Marcos (2018):** A public data set of overground and treadmill walking kinematics and kinetics of healthy individuals. figshare. Dataset. https://doi.org/10.6084/m9.figshare.5722711.v4
4. **Alton F, Baldey L, Caplan S, Morrissey MC. 1998**. A kinematic comparison of overground and treadmill walking. *Clinical Biomechanics* **13**:434-440
5. **Arnold JB, Mackintosh S, Jones S, Thewlis D. 2014**. Differences in foot kinematics between young and older adults during walking. *Gait Posture* **39**:689-694
6. **Dua N, Singh SN, Semwal VB (2021)** Multi-input CNN-GRU based human activity recognition using wearable sensors. Computing. https://doi.org/10.1007/s00607-021-00928-8
7. **Ekinci M (2006)** A new approach for human identification using gait recognition. In: International conference on computational science and its applications. Springer, Berlin
8. **Gupta JP, Polytool D, Singh N, Semwal VB (2014)** Analysis of gait pattern to recognize the human activities. IJIMAI 2(7):7–16
9. **Gupta A et al (2020)** Multiple task human gait analysis and identification: ensemble learning approach. In: Emotion and information processing. Springer, Berlin
10. **Sivakumar S et al (2016)** ANN for gait estimations: a review on current trends and future applications. In: 2016 IEEE EMBS conference on biomedical engineering and sciences (IECBES). IEEE.
11. **Jason Brownlee,** Stacking Ensemble for Deep Learning Neural Networks in Python https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/
12. **Jason Brownlee**, How to Develop VGG, Inception and ResNet Modules from Scratch in Keras https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/
13. **Aishwarya Singh, June 18, 2018.** How to Develop VGG, Inception and ResNet Modules from Scratch in Keras https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/