

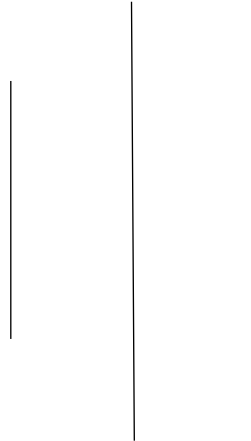


TRIBHUVAN UNIVERSITY

Faculty of management

National College of Computer Studies

Paknajol, Kathmandu



PYTHON REPORT ON MOTION POSE CLASSIFICATION

Submitted by:

Name: Jubisha Prajapati

BIM 5th SEM

Section: B

Roll No:7

Submitted To:

Mausam Rajbanshi

Submission Date _____

Acknowledgement

I would like to extend my heartfelt gratitude for the guidance and support provided by academic and professional mentor, 'Mr. Mausam Rajbanshi' for his constant support and guidance during making this report. His insightful feedback and valuable suggestions were crucial in shaping the structure and content of the report. I am also grateful to my friends whose valuable contributions and feedback helped me to enhance the quality of this report.

I would like to express my appreciation to NCCS college management for giving me this opportunity and providing me with the necessary resources and facilities to carry out this project. Their support was indispensable in ensuring the success of this project. Additionally, I want to thank the scikit-learn library's developers, contributors, and online resources that I have referred to while completing this project. Their contribution has been instrumental in providing us with the necessary information and data to complete this report.

Lastly, I would like to thank my family for their support and encouragement throughout the project process. Their encouragement and belief in my abilities have been a source of motivation and inspiration.

Table of Contents

Introduction.....	1
Machine Learning in python.....	2
□ Background.....	3
Objectives	5
Implementation	6
Conclusion	15

Introduction

In the ever-evolving landscape of computer vision and artificial intelligence, the accurate classification of human motion poses plays a pivotal role in numerous applications, ranging from gesture recognition and human-computer interaction to healthcare and sports analysis. Understanding and categorizing human motion allows machines to comprehend and respond to human actions, fostering innovation in areas such as robotics, gaming, and healthcare monitoring.

Motion pose classification is the process of recognizing and categorizing specific human body poses or movements from data, such as images, videos, or sensor readings. It is a fundamental task in computer vision and machine learning that involves identifying and assigning a label or class to a given human pose or motion based on a set of predefined categories. The primary goal of motion pose classification is to teach a computer system to understand and interpret human movements,

This report delves into the exciting field of motion pose classification, showcasing the utilization of Python as a powerful tool for developing sophisticated models and algorithms. We will explore the fundamental concepts, challenges, and techniques involved in classifying human motion poses, with a focus on how Python, coupled with popular libraries and frameworks, enables us to build robust and accurate motion pose classification systems.

Machine learning in python

Machine learning in Python is a popular and powerful combination for developing and implementing machine learning models and algorithms. Python's simplicity, readability, extensive libraries, and vibrant community make it an ideal choice for working on machine learning projects. Python's rich library ecosystem, combined with its readability and ease of use, makes it an excellent choice for data scientists looking to effectively explore, analyze, and model data. Depending on the task, you can use a variety of libraries and tools to achieve your data science objectives.

Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computer systems to improve their performance on a specific task through learning from data, without being explicitly programmed. In other words, machine learning allows computers to learn from experience or examples and make predictions or decisions based on that learned knowledge.



Here are some key concepts and components of machine learning:

- i. **Data:** Machine learning heavily relies on data. It requires a substantial amount of data to train models effectively. This data can come in various forms, including text, images, numerical values, and more.
- ii. **Algorithms:** Machine learning algorithms are the mathematical and computational techniques used to discover patterns, relationships, and insights from data. These algorithms can be categorized into different types, including supervised learning, unsupervised learning, and reinforcement learning.
- iii. **Model Training:** To train a machine learning model, you provide it with a dataset and use an algorithm to adjust the model's parameters so that it can make accurate predictions or decisions. The training process involves optimizing the model's performance on the given task.

- iv. **Evaluation:** After training, it's crucial to assess the model's performance to ensure it can generalize well to unseen data. Common evaluation metrics depend on the specific task but may include accuracy, precision, recall, F1-score, mean squared error (MSE), and many others.
- v. **Deployment:** Once a machine learning model has been trained and evaluated, it can be deployed in real-world applications to make predictions or automate decision-making. Deployment often involves integrating the model into software or systems.

- **Background**

Classification is one of the fundamental tasks in machine learning, and it refers to the process of categorizing or labelling data into distinct classes or categories based on certain features or attributes. Common algorithms for classification in Python include:

- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- k-Nearest Neighbors (k-NN)
- Naive Bayes
- Neural Networks (Deep Learning)

Python provides various libraries and frameworks, such as scikit-learn, TensorFlow, and PyTorch, to implement classification algorithms effectively.

Data classification is a fundamental task in machine learning that involves training a model to classify data into predefined classes or categories. The motion pose dataset includes collection of data containing recorded information about human or object poses over time.

The primary goal of this data classification task is to construct a machine learning model capable of accurately classifying Iris flowers into their respective species based on these four attributes. To accomplish this, we use Python, a popular programming language for data analysis and machine learning, and a variety of libraries and techniques to handle the dataset, build a classification model, and evaluate its performance.

The following are the general steps involved in this data classification task:

- Data Collection: To begin, we obtain the motion pose dataset, which is easily accessible in Python via the scikit-learn library.
- Data Preparation: To facilitate data manipulation, the dataset is organized into a structured format, typically a Data Frame
- Algorithm Selection: We select an appropriate machine learning algorithm for this classification task. In this example, we use the decision tree algorithm and logistic regression algorithm, which is a straightforward yet effective method for

classification tasks. The algorithm chosen is determined by the nature of the data and the requirements of the problem.

- Model Training: Using the training dataset, the selected algorithm is trained. The model learns to recognize patterns and relationships in the data during this phase, allowing it to make predictions.
- Model Evaluation: The model is evaluated using the testing dataset after it has been trained. To evaluate the model's performance, metrics such as accuracy, precision, recall, and the F1-score are calculated. A confusion matrix is also generated to visualize the model's classification results.
- Interpretation and Reporting: Model evaluation results are interpreted and reported on to determine the model's accuracy and effectiveness in classifying motion poses. These findings are then documented, most commonly in the form of a classification report and a confusion matrix.

This basic background gives an overview of the steps required to perform data classification on the motion pose dataset using Python. The process is invaluable for understanding the principles of data classification and model evaluation and serves as a foundational example for machine learning practitioners.

Objectives

Here are some of the objectives of making this report: -

- To use Python to perform data classification on the motion pose dataset.
- To analyse and classify motion pose using machine learning algorithms.
- To identify the correlations between the parameters.
- To document the entire process of motion pose classification, from data collection and pre-processing to model development and evaluation.
- To share knowledge and insights gained during the project with a wider audience, including colleagues, researchers etc.
- To improve result visualization, visual aids such as a confusion matrix.
- To objectively evaluate the performance of the classification model and provide a clear assessment of its accuracy, precision, and other relevant metrics.
- To go over the reasoning behind data cleaning, dealing with missing values, and standardizing features.

Implementation

To carry out the classification task, we use Python, a versatile programming language for data analysis and machine learning. The following steps outline the key implementation elements:

1. Load Data:

The motion pose dataset is loaded to begin the implementation. This dataset is easily accessible in Python via the scikit-learn library, which is a powerful machine learning tool. The four CSV files, `x_test`, `x_train`, `y_test` and `y_train` are loaded into Python environment.

2. Prepare data:

Preprocess the data as needed. This may include handling missing values, encoding categorical variables, and scaling or normalizing features.

3. Merge or Concatenate Datasets:

If the four CSV files represent different aspects of the same dataset or are meant to be combined, merge or concatenate them into a single data frame. For this purpose, `x_train` and `y_train` data have been concatenated and placed in one variable i.e. `train_data`.

4. Data Exploration and Analysis:

The combined dataset is explored to understand its characteristics, distributions, and relationships between variables. Pandas are used for data exploration and libraries like Seaborn for data visualization.

5. Split the Data:

We divide the dataset into two subsets to facilitate model training and evaluation: a training set and a testing set. The training set is used to train your classification model, while the testing set is used to evaluate its performance. In this implementation, 80% of the data is allocated to the training set, with the remaining 20% used to test the model's performance. This division ensures that the model learns from a subset of the data and is tested on an independent dataset to determine its ability to generalize. `train_test_split()` function from scikit-learn is used for this purpose.

6. Choose a Classification Model and Model Training:

We select an appropriate machine learning classification model based on the nature of our data. Common choices include logistic regression, decision trees, random forests, and deep learning models like neural networks. Here, we have used logistic regression and decision tree.

7. Model Evaluation:

Here, we evaluate the performance of your classification model using appropriate evaluation metrics such as accuracy and confusion matrices. This helps you assess how well your model classifies data. We include visual aids such as a confusion matrix to improve the presentation of results and use heatmap to present the correlation in figure.

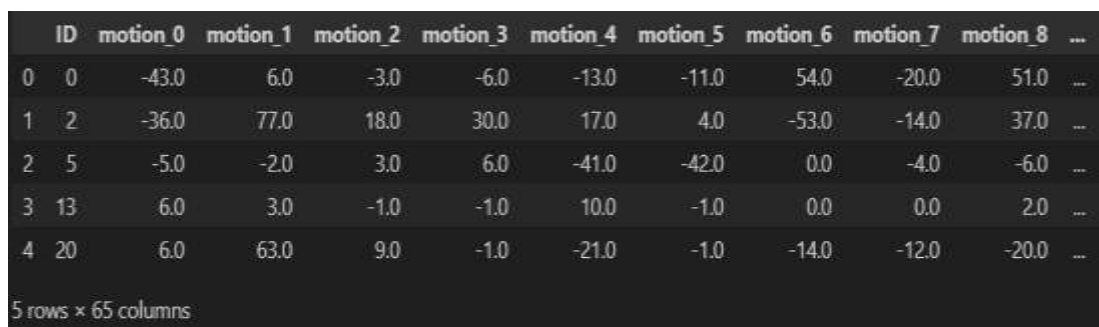
The codes used for this project are given below:

- **Necessary imports:**

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

- **Read the CSV file:**

```
x_test = pd.read_csv("./csv_files/x_test.csv")
x_test.head()
```



	ID	motion_0	motion_1	motion_2	motion_3	motion_4	motion_5	motion_6	motion_7	motion_8	...
0	0	-43.0	6.0	-3.0	-6.0	-13.0	-11.0	54.0	-20.0	51.0	...
1	2	-36.0	77.0	18.0	30.0	17.0	4.0	-53.0	-14.0	37.0	...
2	5	-5.0	-2.0	3.0	6.0	-41.0	-42.0	0.0	-4.0	-6.0	...
3	13	6.0	3.0	-1.0	-1.0	10.0	-1.0	0.0	0.0	2.0	...
4	20	6.0	63.0	9.0	-1.0	-21.0	-1.0	-14.0	-12.0	-20.0	...

5 rows x 65 columns

```
x_train = pd.read_csv("./csv_files/x_train.csv")
x_train.head()
```

	ID	motion_0	motion_1	motion_2	motion_3	motion_4	motion_5	motion_6	motion_7	motion_8	...
0	1	-6.0	-1.0	-4.0	-15.0	9.0	13.0	-65.0	-19.0	-25.0	...
1	3	0.0	10.0	5.0	21.0	2.0	-8.0	15.0	19.0	-12.0	...
2	4	16.0	5.0	6.0	10.0	-12.0	-27.0	-4.0	-3.0	-13.0	...
3	6	8.0	-16.0	-13.0	2.0	-8.0	-12.0	-54.0	-7.0	-10.0	...
4	7	13.0	-1.0	0.0	14.0	-8.0	-10.0	-78.0	-4.0	-9.0	...

5 rows × 65 columns

```
y_test = pd.read_csv("./csv_files/y_test.csv")
```

```
y_test.head()
```

	ID	pose
0	0	0
1	2	0
2	5	1
3	13	1
4	20	0

```
y_train = pd.read_csv("./csv_files/y_train.csv")
```

```
y_train.head()
```

	ID	pose
0	1	0
1	3	0
2	4	1
3	6	0
4	7	0

- Combining the data

```
train_data = x_train.copy()
train_data['pose'] = y_train['pose']
train_data
```

	ID	motion_0	motion_1	motion_2	motion_3	motion_4	motion_5	motion_6	motion_7	motion_8	...
0	1	-6.0	-1.0	-4.0	-15.0	9.0	13.0	-65.0	-19.0	-25.0	...
1	3	0.0	10.0	5.0	21.0	2.0	-8.0	15.0	19.0	-12.0	...
2	4	16.0	5.0	6.0	10.0	-12.0	-27.0	-4.0	-3.0	-13.0	...
3	6	8.0	-16.0	-13.0	2.0	-8.0	-12.0	-54.0	-7.0	-10.0	...
4	7	13.0	-1.0	0.0	14.0	-8.0	-10.0	-78.0	-4.0	-9.0	...
...
4645	5805	-22.0	-3.0	-2.0	-1.0	-6.0	-3.0	-7.0	-18.0	15.0	...
4646	5806	6.0	-3.0	-11.0	-8.0	-10.0	-11.0	-55.0	-21.0	-19.0	...
4647	5807	-38.0	2.0	3.0	2.0	-15.0	9.0	0.0	-11.0	67.0	...
4648	5811	22.0	7.0	-4.0	-2.0	1.0	32.0	-2.0	-5.0	-4.0	...
4649	5812	9.0	5.0	-11.0	-37.0	13.0	36.0	-10.0	-27.0	-30.0	...

4650 rows × 66 columns

- Describing the data

```
train_data.describe()
```

	ID	motion_0	motion_1	motion_2	motion_3	motion_4	motion_5
count	4650.000000	4650.000000	4650.000000	4650.000000	4650.000000	4650.000000	4650.000000
mean	2911.594194	-0.180215	-0.637419	-0.720000	-0.724946	-0.214194	-0.354839
std	1679.082434	21.789717	9.110501	4.862874	8.172864	18.424552	20.891347
min	1.000000	-116.000000	-99.000000	-27.000000	-65.000000	-121.000000	-95.000000
25%	1456.250000	-11.000000	-3.000000	-3.000000	-4.000000	-10.000000	-11.000000
50%	2917.000000	-2.000000	-1.000000	-1.000000	-1.000000	0.000000	-1.000000
75%	4363.750000	8.000000	2.000000	2.000000	3.000000	10.000000	11.000000
max	5812.000000	111.000000	85.000000	31.000000	43.000000	81.000000	105.000000

8 rows × 66 columns

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4650 entries, 0 to 4649
Data columns (total 66 columns):
#   Column      Non-Null Count  Dtype  
---  -
0   ID           4650 non-null   int64  
1   motion_0     4650 non-null   float64
2   motion_1     4650 non-null   float64
3   motion_2     4650 non-null   float64
4   motion_3     4650 non-null   float64
5   motion_4     4650 non-null   float64
6   motion_5     4650 non-null   float64
7   motion_6     4650 non-null   float64
8   motion_7     4650 non-null   float64
9   motion_8     4650 non-null   float64
10  motion_9     4650 non-null   float64
11  motion_10    4650 non-null   float64
12  motion_11    4650 non-null   float64
13  motion_12    4650 non-null   float64
14  motion_13    4650 non-null   float64
15  motion_14    4650 non-null   float64
16  motion_15    4650 non-null   float64
17  motion_16    4650 non-null   float64
18  motion_17    4650 non-null   float64
19  motion_18    4650 non-null   float64
...
64  motion_63    4650 non-null   float64
65  pose         4650 non-null   int64  
dtypes: float64(64), int64(2)
memory usage: 2.3 MB
```

- **Check for duplicated and null values:**

```
train_data.isnull().sum()
```

```
ID          0
motion_0     0
motion_1     0
motion_2     0
motion_3     0
..
motion_60    0
motion_61    0
motion_62    0
motion_63    0
pose         0
Length: 66, dtype: int64
```

```
train_data.duplicated().sum()
```

```
0
```

- **Correlation matrix**

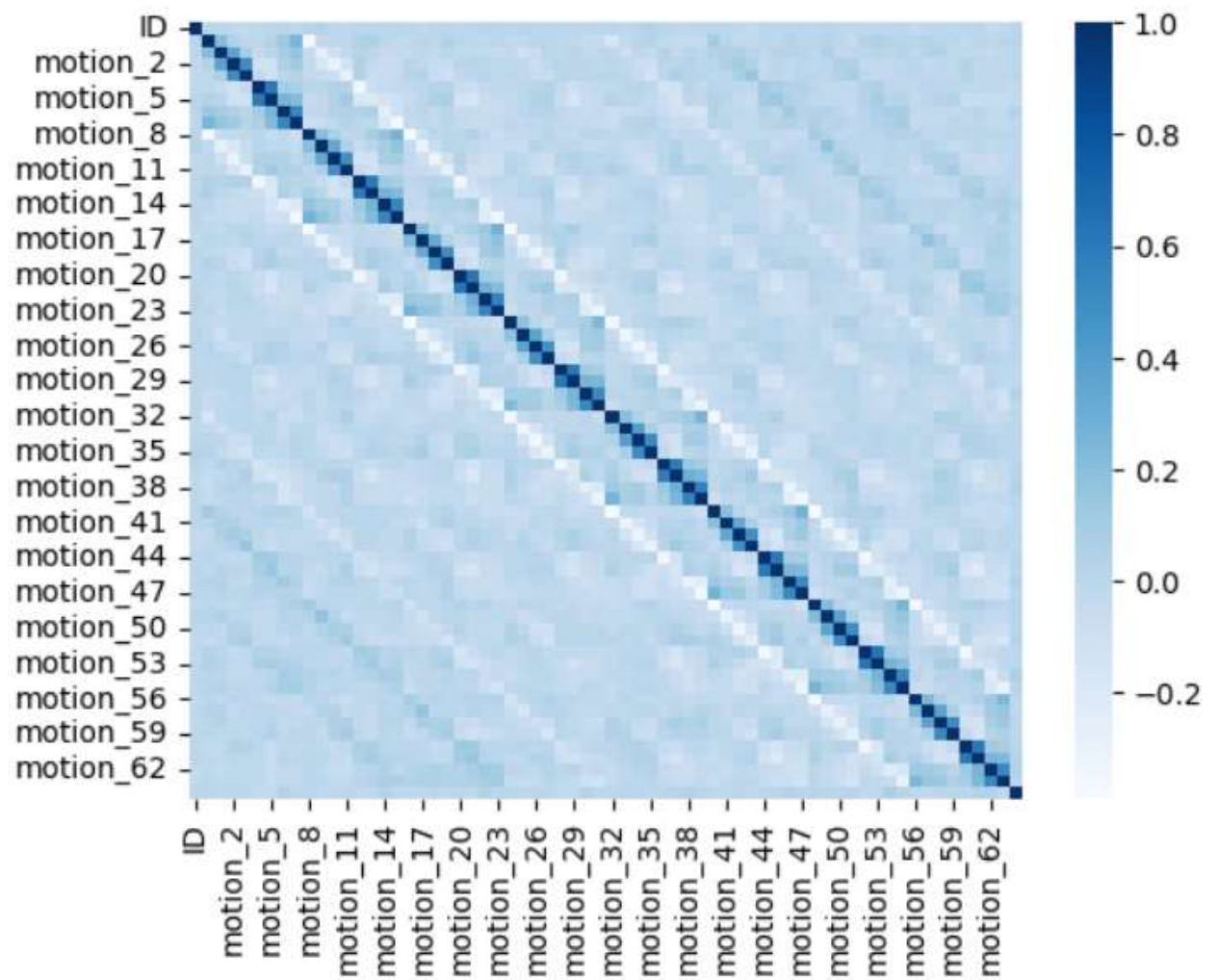
```
train_data.corr()
```

	ID	motion_0	motion_1	motion_2	motion_3	motion_4	motion_5	motion_6	motion_7	motion_8	...
ID	1.000000	0.013121	0.000457	-0.019301	-0.028087	-0.004864	0.010296	0.006359	-0.013266	0.011694	...
motion_0	0.013121	1.000000	0.211825	0.034156	-0.011007	-0.005725	0.042003	0.141925	0.251372	-0.370308	...
motion_1	0.000457	0.211825	1.000000	0.371688	0.075426	-0.047240	0.009861	0.101261	0.162220	-0.037811	...
motion_2	-0.019301	0.034156	0.371688	1.000000	0.501969	-0.116049	-0.105428	0.064594	0.118102	-0.030974	...
motion_3	-0.028087	-0.011007	0.075426	0.501969	1.000000	-0.028308	-0.110293	0.002942	0.025969	-0.017048	...
...
motion_60	-0.002124	-0.022546	-0.017051	0.040446	0.028413	-0.008071	-0.062107	-0.015784	-0.013102	0.000645	...
motion_61	0.001200	-0.003415	-0.020286	-0.003482	-0.009440	-0.011614	-0.059104	0.008692	-0.003116	0.013365	...
motion_62	-0.027220	-0.007072	-0.034287	-0.005406	-0.005989	-0.007252	-0.054801	-0.022895	-0.016034	0.011988	...
motion_63	-0.026285	0.014861	-0.020650	-0.005036	-0.006439	-0.019020	-0.050621	-0.024718	-0.031125	0.011691	...
pose	0.015071	0.016355	0.004159	0.001313	0.005754	-0.025206	-0.029919	0.033473	-0.001311	0.000487	...

66 rows x 66 columns

```
import seaborn as sns
```

```
sns.heatmap(train_data.corr(), cmap='Blues')
```



- **Model Training**

```
x=train_data.drop(['pose'],axis=1)
y=train_data['pose']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size= 0.20, random_state=0)
```

- **Logistic Regression**

```
from sklearn.linear_model import LogisticRegression
```

```
classify_model = LogisticRegression(max_iter=1000)
```

```
classify_model.fit(X_train, y_train)
```

```
classify_model
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
preds = classify_model.predict(X_test)
```

```
preds
```

```
array([1, 0, 0, ..., 1, 1, 0], dtype=int64)
```

- **Confusion matrix**

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test,preds)
```

```
array([[ 884, 1005],
       [ 425, 1406]], dtype=int64)
```

```
accuracy_score(y_test,preds)
```



```
0.6155913978494624
```

- **Decision Tree**

```
from sklearn.tree import DecisionTreeClassifier  
dt_classifier = DecisionTreeClassifier()  
dt_classifier.fit(X_train, y_train)  
dt_classifier.predict(X_test)
```

```
array([1, 1, 0, ..., 1, 1, 1], dtype=int64)
```

```
preds_dt = dt_classifier.predict(X_test)  
print(preds_dt)
```

```
print(confusion_matrix(y_test, preds_dt))
```

```
[[1855  34]  
 [  73 1758]]
```

```
print(accuracy_score(y_test, preds_dt))
```

```
0.9712365591397849
```

Conclusion

In conclusion, I embarked on a journey of data classification using Python in this report, focusing on the well-known motion pose dataset. This report has explored the exciting and versatile realm of motion pose estimation using Python. We have delved into various aspects of this field, including the importance of motion pose analysis, the techniques and algorithms commonly employed.

Motion pose estimation plays a crucial role in a wide range of applications, from sports analytics to healthcare and robotics. It enables us to extract valuable insights from motion data, enhance human-computer interactions, and improve the efficiency and safety of various processes.

Python, with its extensive libraries and frameworks, offers a powerful platform for developing motion pose estimation solutions. Throughout this report, we have discussed key concepts such as data preprocessing, model training etc. Additionally, we have highlighted the importance of data quality and model selection in achieving accurate results. The proposed deep learning model was implemented using the Python programming language with packages and libraries. The motion pose dataset was used to perform basic data analysis, then data standardization and visualization were performed. Finally, the model was trained for the accuracy and performance of the model were also evaluated.

As technology continues to advance, motion pose estimation will only become more relevant and impactful. In conclusion, I gained valuable insights from this project and came to know that motion pose estimation using Python is a dynamic and evolving field with a broad range of applications, and it holds the promise of transforming how we interact with and understand the world around us.