National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Krishnanunni Raju
Student ID: 20232217

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Krishnanunni Raju
Student ID:	20232217
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Mr. Hicham Rifai
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1642
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Krishnanunni Raju
20232217

1 Introduction

The goal of this document is to offer all of the instructions required to reproduce the Exercise Tracking and Detection Using Spatial-Temporal Graph Convolutional Neural Network implementation.

2 Hardware Requirements

The computer used for project implementation has a 9^{th} generation Intel Core i7-9750H@2.60 GHz processor, 16GB of RAM, and Microsoft Windows 11 Home edition. The hardware specification is given in Figure 1.

Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Device ID	E23DA98D-2237-47AF-BC81-C36A8039A3B1
Product ID	00327-35849-64710-AAOEM
System type	64-bit operating system, x64-based processor

Figure 1: Hardware Specification

The specifications of the operating system is shown in Figure 2.

OS Name	Microsoft Windows 11 Home Single Language
Version	10.0.22000 Build 22000
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation

Figure 2: Operating system specification

3 Software Requirements

Pycharm Professional Edition by JetBrains was used to implement the solution. The version of the software is 2022.1.2. The software specification is shown in Figure 3. Python 3.10.4 is used for implementation as shown in Figure 4.

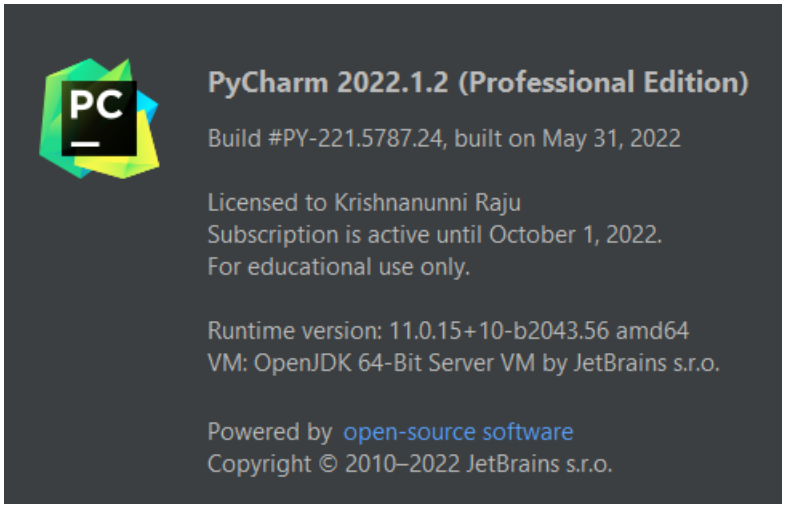

Figure 3: Pycharm software specification

```
C:\Users\user>py --version
Python 3.10.4
```

Figure 4: Python version

4 Library requirements

The following are the libraries required and their versions. Pip package manager is used to install all these libraries.

- mediapipe - 0.8.10
- numpy - 1.23.0
- opencv_contrib_python - 4.6.0.66
- pandas - 1.4.3
- torch - 1.12.0
- protobuf - 3.19.0

5 Dataset Description

- The dataset used for implementation is an open-source dataset. It is downloaded from InfinityAI website[1]. It is also available to be downloaded from another source[2].
- The dataset is in the form of a zip file. Once extracted there will be 10 zip files with each file having 100 videos in it as shown in Figure 5. These files should be extracted and a folder structure should be created as shown in Figure 6.

[1]https://toinfinity.ai/infiniterep
[2]https://paperswithcode.com/dataset/infiniterep

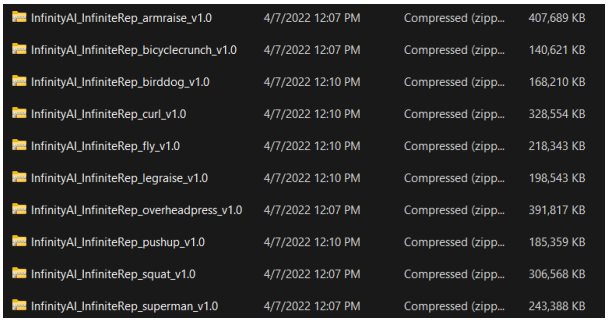

Figure 5: Extracted dataset

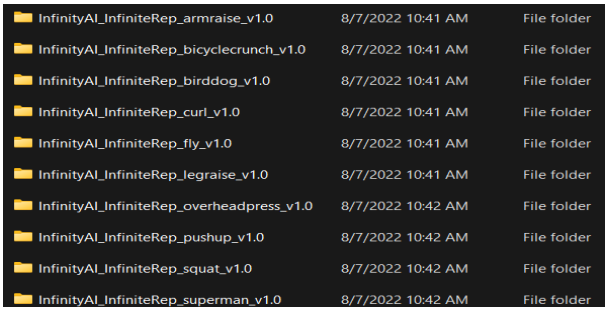

Figure 6: Folder structure of the data

6 Training Data Generation

- Once the dataset folder structure is created class TrainingData is utilized to create the training data.
- The constructor of the class takes the path of dataset folder.
- TrainingData class has three member functions. They are $generate\_frames()$, $read\_files()$ and $read\_video\_files\_into\_dataframe()$. These functions are internally used by the class and called inside the constructor.
- All the directories are looped and files with extension $mp4$ are read.
- An object of PoseEstimator class is created and the skeletal information is extracted using the $capture\_from\_training\_data()$ method.

7 Pose Capturing

PoseEstimator is the class used to capture the skeletal information. The class diagram is shown in Figure 7

- It has two important memeber functions. They are $capture()$ and $capture\_from\_training\_data()$.
- $capture()$ is used to extract skeletal data and classify the exercise in real time.
- $capture\_from\_training\_data()$ helps the TrainingData class to extract skeletal data from the video dataset.

- There are three more functions in the class that are internally used by the other two functions for achieving their goal.

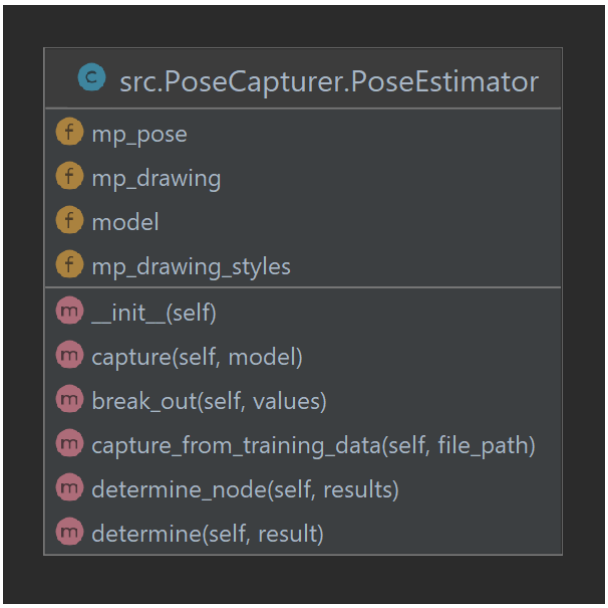

Figure 7: PoseEstimator Class

8 Model Implementation

8.1 STGCN Class

- The model is implemented using the Model class and class STGCN acts as a wrapper of the Model class. The fields and methods of the class are shown in Figure 8
- Arguments of STGCN class:
  - optimizer(string): The optimizer to be considered for training.
  - labels(list): List of labels related to the network.
  - strategy(string): The partitioning method to be used. This can take two values either 'spatial' or 'uniform'. If any other values are provided a ValueError exception will be raised.
  - edge_importance(bool): This parameter decides whether the learnable edge importance mask will be used or not.
- Functionality of STGCN class:
  - Create an instance of Model class. Model class is the implementation of the Spatial-temporal Graph Convolutional Neural Network.
  - Train the STGCN model using *train*( ) function. The function takes two arguments. They are the path of skeletal data in .NPY file and labels in a pickle file.

- The number of epochs can be changed inside the loop defined in *train*( ) method. It is set to 100.
- Batch size for training can be changed in the arguments while initializing *loader* variable. It is set to 256 for training.
- Test the model with a testing data using *test*( ). It takes a single argument for the path of data. The data should be stored in .NPY file format.
- Batch size for test data could be changed by changing the *batch_size* argument value while initializing the *loader* variable
- Predict the class of exercise for any given data. The *predict*( ) function used for this takes the data as the argument. The data should have the form a tensor.
- *save*( ) takes two arguments. They are the path where the model should be saved and path where the labels should be saved.
- *load*( ) takes path of the saved model and path of the labels as arguments. It loads the saved model.

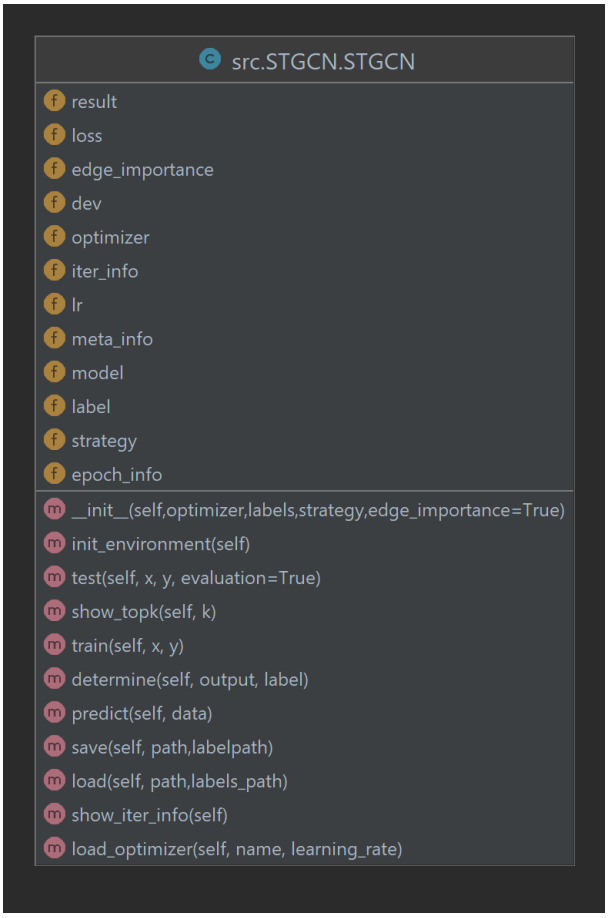

Figure 8: STGCN Class

8.2 Model Class

- Arguments of the Model class:

- in_channels (int): The number of coordinates or channels in the data. It is set to 3.
- num_class (int): The count of classes of exercises. Number of different exercises that will be detected.
- edge_importance_weighting (bool): If it is set to $True$, then a learnable importance weighting is added to the edges of the graph.

- Functionality of Model class:

  - The neural network is defined in this class. The constructor of the class is shown in Figure 9

```
def __init__(self, in_channels, num_class,
             edge_importance_weighting,strategy, **kwargs):
    super().__init__()

    # load graph
    self.graph = Graph(strategy=strategy)
    A = torch.tensor(self.graph.A, dtype=torch.float32, requires_grad=False)
    self.register_buffer('A', A)

    # build networks
    spatial_kernel_size = A.size(0)
    temporal_kernel_size = 9
    kernel_size = (temporal_kernel_size, spatial_kernel_size)
    self.data_bn = nn.BatchNorm1d(in_channels * A.size(1))
    kwargs0 = {k: v for k, v in kwargs.items() if k != 'dropout'}
    self.st_gcn_networks = nn.ModuleList((
        st_gcn(in_channels, 64, kernel_size, 1, residual=False, **kwargs0),
        st_gcn(64, 64, kernel_size, 1, **kwargs),
        st_gcn(64, 64, kernel_size, 1, **kwargs),
        st_gcn(64, 64, kernel_size, 1, **kwargs),
        st_gcn(64, 128, kernel_size, 2, **kwargs),
        st_gcn(128, 128, kernel_size, 1, **kwargs),
        st_gcn(128, 128, kernel_size, 1, **kwargs),
        st_gcn(128, 256, kernel_size, 2, **kwargs),
        st_gcn(256, 256, kernel_size, 1, **kwargs),
        st_gcn(256, 256, kernel_size, 1, **kwargs),
    ))
```

Figure 9: Constructor of Model Class

  - Model class is inherited from $torch.nn.Module$ class.
  - Define the number of ST-GCN blocks with the required arguments.
  - Function $forward(\,)$ takes the data in the form of tensor and performs forward propagation.
  - The fields and methods of the class are shown in Figure 10

8.3 st-gcn class

- Applies a spatial temporal graph convolution over an input graph sequence.
- It is also inhereted from $torch.nn.Module$. The fields and methods of the class are shown in Figure 11.
- Arguments of st-gcn class
  - in_channels (int): Number of coordinates/channels in the input data.
  - out_channels (int): Count of channels after the convolution operation.

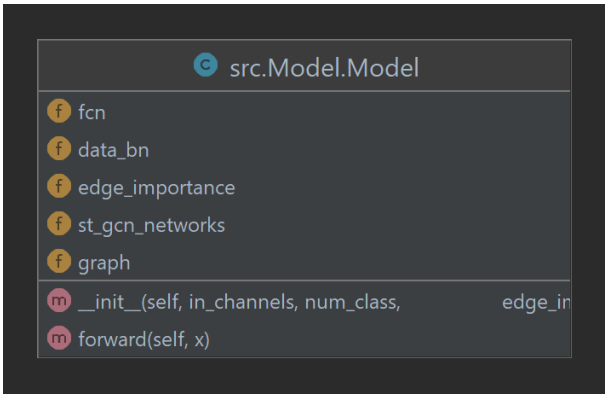

Figure 10: Model Class

- kernel_size (tuple): It has a datatype of tuple. The first value corresponds to the size of temporal convolving kernal and the second value corresponds to the size of graph convolvong kernal.
- stride (int): The default value of stride is 1 and it denotes the stride of the temporal convolution operation. As the default value is set it is a optional argument.
- dropout (int): Defines the value of hyperparameter dropout. The default value is set to 0. It is an optional argument.
- residual (bool): If it is set to $True$, it applies a residual mechanism.The default value is set to $True$.

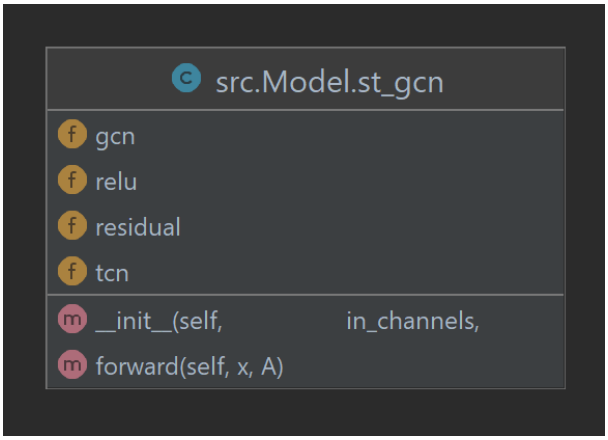

Figure 11: st_gcn class

- Shape:
  - Input[0]:The sequence of input graph(Format: $(N, in\_channels, T\_in, V)$)
  - Input[1]: The adjacency matrix of the input graph(Format: (K, V, V))
  - Output[0]: Defines the output graph sequence. (Format: $(N, out\_channels, T\_out, V)$)
  - Output[1]: Defines the adjacency matrix of the graph for output data. (Format: (K, V, V)).
    Here

N - batch size.
K - spatial kernel size.
T_in/T_out - length of the input or output sequence.
V - number of nodes int the graph.

8.4 Graph Class

- This class defines the graph used to model the skeletons that are extracted by MediaPipe BlazePose. The fields and methods of the class is shown in Figure 12
- Arguments of Graph Class:
  - strategy (string): Defines the partitioning strategy used. It can only be one of the two following values.
    1. uniform: Uniform Labeling
    2. spatial: Spatial Configuration
  - max_hop (int): the maximum distance in between two of the connected nodes.
  - dilation (int): Defines the spacing in between the kernel points.

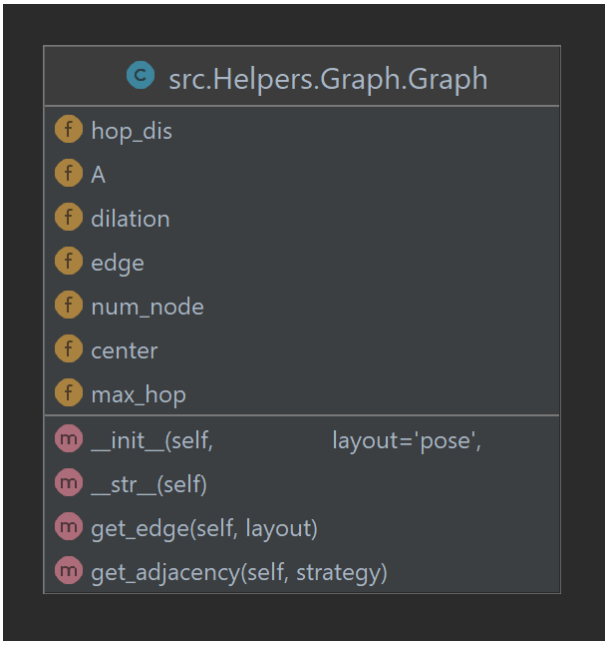

Figure 12: Graph class

8.5 ConvTemporalGraphical Class

- This class is also inherited from *torch.nn.Module*.
- This is the class for application of a graph convolution. The fields and methods of this class is shown in Figure 13.
  - in_channels (int): Defines the count of coordinates/channels in the input data.

- out_channels (int): Number of channels that are present after the convolution operation.
- kernel_size (int): Defines the size kernel that performs graph convolution.
- t_kernel_size (int): Defines the size of the kernel that performs the temporal convolution.
- t_stride (int): It is a optional parameter and default value is 1. It defines the stride value for temporal convolution.
- t_padding (int): Temporal zero-padding that are added to both the sides of input. It is an optional parameter and default value is 0.
- t_dilation (int):Defines the spacing in between the elements in temporal kernel. It is an optional parameter and default value is 1.
- bias (bool): If it is set to $True$, a learnable bias is added to the output. It is an optional argument and the default value is $True$.

Figure 13: ConvTemporalGraphical Class

8.6 Feeder Class

- Data feeder class for model training and testing. The fields and methods in the class is shown in Figure 14.
- Arguments for Feeder Class:
  - data_path: the path to '.npy' data, the shape of data should be (N, C, T, V, M)
  - label_path: the path to label
  - random_choose: If true, randomly choose a portion of the input sequence
  - random_shift: If true, randomly pad zeros at the beginning or end of sequence
  - window_size: The length of the output sequence
  - normalization: If true, normalize input sequence debug: If true, only use the first 100 samples

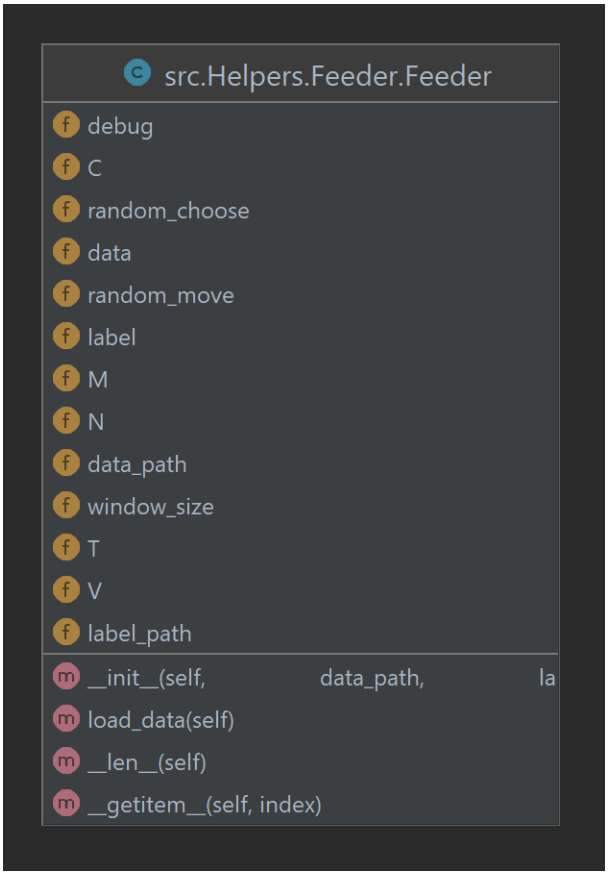

Figure 14: Feeder class

8.7 ExerciseTracker class

- The whole implementation follows compositional architecture. Exercise Tracker is the class that uses all the other classes and does model training, testing and real-time tracking as well. The fields and methods of the class are shown in Figure 15.
- Arguments for ExerciseTracker:
  - train(bool): If set to True the models will be trained and if set to False will try to load a saved model.
  - strategy(string): Can take two values: 'spatial' or 'uniform' depending on the strategy required. Only applicable for loading the model.
  - edge_importance(bool): Only applicable if loading a saved model. If set to $True$, it will try to load the model with edge importance set to $True$.

9 Training

In *main.py* file call ExerciseTracker with train parameter set to $True$. The other two parameters are not required if training is to be performed as shown in Figure 16.

This will train 4 models. They are created with a combination of two partitioning methods and use of learnable edge importance weighting mask. It will also test the models with a testing data and print the results. If $track()$ function is called after this

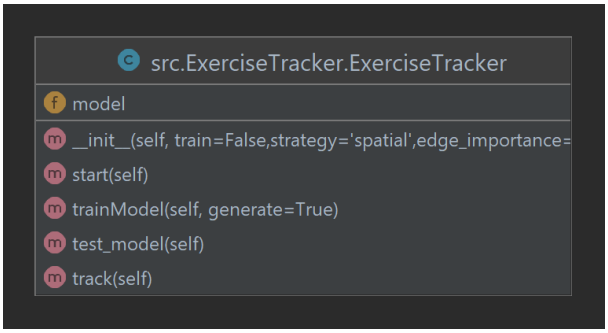

Figure 15: ExerciseTracker Class

```
from src.ExerciseTracker import ExerciseTracker

if __name__ == '__main__':
    instance=ExerciseTracker(train=True)
```

Figure 16: Configuring main.py for training and saving the model

line it will access the camera and start the detection.

Figure 17 shows the functions *trainModel*( ),*test_model*( ) and *track*( ). If *generate* is set

```
def trainModel(self, generate=True):
    if generate:
        strategy=['uniform','spatial']
        edge_importance=[False,True]
        training_data=TrainingData('C:\Project DBs\Final Research DB')
        for strat in strategy:
            for ei in edge_importance:
                self.model = STGCN('Adam',training_data.labels,strategy=strat,edge_importance=ei)
                self.model.train(x="C:\Project DBs\Final Research DB\\final_data.npy",y='C:\Project DBs\Final Research DB\\final_data_label.pkl')
                self.model.save(f'C:\Project DBs\Final Research DB\\Final_Model_{strat}_{ei}.pth',f'C:\Project DBs\Final Research DB\\Final_Model_
                 {strat}_{ei}_Label.pkl')
                self.test_model()

def test_model(self):
    self.model.test(x="C:\Project DBs\Final Research DB\\test_data.npy",
                    y='C:\Project DBs\Final Research DB\\test_data_label.pkl')

def track(self):
    pose_estimator = PoseEstimator()
    pose_estimator.capture(self.model)
```

Figure 17: Functions used to train the model, test the model with test data and start tracking using the model

to True it will generate training data from videos. The argument to TrainingData should be the directory where the extracted dataset is present as shown in Figure 6 in section 5. If the folder path has been changed, change the argument to TrainingData. It will save the data and labels in $C:\backslash Project\ DBs\backslash Final\ Research\ DB\backslash final\_data.npy$ and $C:\backslash Project\ DBs\backslash Final\ Research\ DB\backslash final\_data\_label.pkl$ respectively. The models will be saved in $C:\backslash Project\ DBs\backslash Final\ Research\ DB$.

```
from src.ExerciseTracker import ExerciseTracker

if __name__ == '__main__':
    instance=ExerciseTracker(train=False,strategy='spatial',edge_importance=True)
    instance.track()
```

Figure 18: Configuring main.py for using a saved model

10 Using a saved model

To use a saved model initialize ExerciseTracker with *train* as *False*. Provide arguments *strategy* and *edge_importance* for loading the required model. If *strategy* is given as 'spatial' and *edge_importance* is set to *True* it will load model with spatial configuration partitioning and learnable edge importance weighting mask.
Call *track*( ) method of ExerciseTracker to access the camera and start the detection and classification.

```
def __init__(self, train=False,strategy='spatial',edge_importance=True):
    print('Initialized')
    if train:
        self.start()
        self.model = self.trainModel()
    else:
        with open(f'C:\Project DBs\Final Research DB\\Final_Model_{strategy}_{edge_importance}_Label.pkl','rb') as f:
            labels=pickle.load(f)
            self.model = STGCN('Adam',list(labels),strategy=strategy)
        self.model.load(f'C:\Project DBs\Final Research DB\\Final_Model_{strategy}_{edge_importance}.pth',f'C:\Project DBs\Final Research
         DB\\Final_Model_{strategy}_{edge_importance}_Label.pkl')
```

Figure 19: ExerciseTracker constructor

From the constructor of ExerciseTracker shown in Figure 19 it could be seen that the saved model will be looked in $C : \backslash Project\, DBs \backslash Final\, Research\, DB$. The name of the model file will be in the format Final_Model_{strategy name}_{edge importance weighting enabled}.pth.