Configuration Manual

Research Project
MSc.Data Analytics

Aarthi Rajendran
Student ID: 22224947

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……..Aarthi Rajendran…………………………………………………………………………

Student ID: ……22224947……………………………………………………………………………..……

Programme:……Data Analytics……………………………………… **Year:** ………2024……………..

Module: ……Msc Research Project……………………………………………………………….………

Lecturer: ……Ahmed Makki…………………………………………………………………….………

Submission Due Date: ……12/08/2024…………………………………………………………………….………

Project Title: ……Configuration Manual…………………………………………………………….………

Word Count: …………1031…………………… **Page Count:** ………10……………………….…….………

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: ……………Aarthi Rajendran……………………………………………………………………

Date: …………11/08/2024…………………………………………………………………………

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):	

Physical Exercise Pose Detection using BlazePose and Machine Learning Framework

Aarthi Rajendran
22224947

1 Introduction

This manual contains all the details involved in implementation of the research project 'Physical Exercise Pose Detection using BlazePose and Machine Learning Framework'. It begins with an overview of the project's objectives and significance. The second section details the system specifications, including hardware, software, and necessary libraries.
The third part covers the data collection and preprocessing methods, followed by the development and evaluation of machine learning models used for pose detection.
Finally, the manual explains how to utilize BlazePose for detecting exercise poses from images, integrating it with machine learning models for accurate predictions. The conclusion summarizes the results and discusses potential improvements and future directions for the project.

2 Hardware Requirements

2.1 Hardware Specification:

Processor: 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz (base frequency) 2.42 GHz (boost frequency)
Installed RAM: 16.0 GB (15.7 GB usable)
System type 64-bit operating system, x64-based processor
Storage: 500 GB SSD
GPU: 4 GB, Intel(R) UHD Graphics

2.2 Operating System Specification:

OS Name: Microsoft Windows 11 Home Single Language
Version 23H2
Experience Windows Feature Experience Pack 1000.22700.1020.0

3 Software Requirements

Programming Language: Python
Cloud Platform: Google Drive for Data Storage
IDE: Google Colab
Other Tools: Microsoft Excel and PowerPoint

4 Library Requirements

To work with physical activity pose detection, Google Colab is a cloud-based platform accessed via a web browser. After signing in, by creating a new notebook or uploading the file. it is important to use specific machine learning libraries. MediaPipe provides advanced capabilities for detecting and analyzing physical activity poses. H2O provides a comprehensive suite of machine learning tools and algorithms for predictive modeling and data analysis. This setup is essential for predicting and evaluating the strength training exercise poses and for performing sophisticated predictive modeling tasks.

```
%pip install -q mediapipe
%pip install h2o
!wget -O pose_landmarker.task -q https://storage.googleapis.com/mediapipe-models/pose_landmarker/pose_landmarker_heavy/float16/1/pose_landmarker_heavy.task
```

Figure 1: MediaPipe and H2O Installation

```
import mediapipe as mp
import cv2
import pandas as pd
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
import os
import h2o
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import cycle
```

Figure 2: All Required Python libraries and Packages

5 Dataset Description

5.1 Dataset selection

Dataset is downloaded from the UCF 101 Human Action Recognition data which are collected from youtube and have a fixed rate frame.

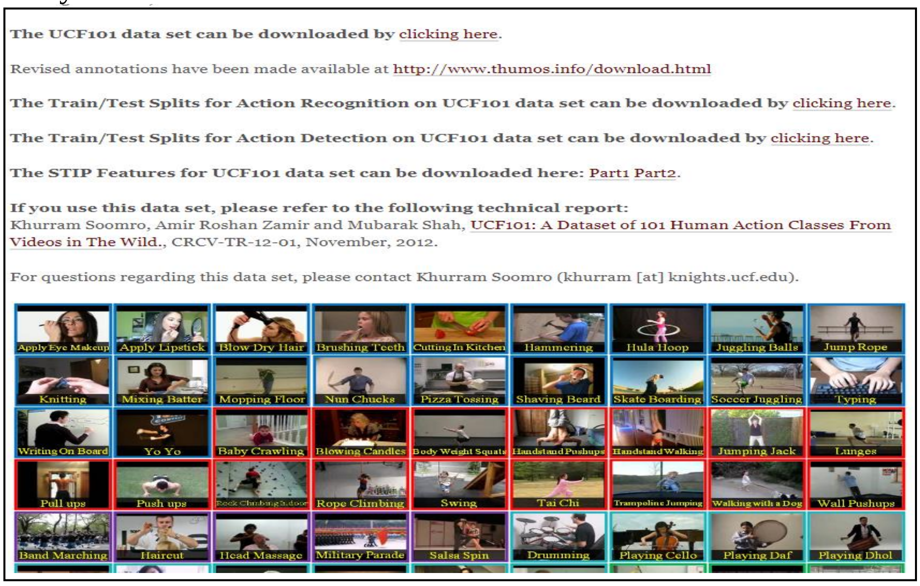

Figure 3: Dataset

5.2 Dataset Collection and Loading

The dataset is downloaded and manually five strength training videos are filtered and uploaded to the Google Drive and accessed by mounting Google Drive in the Colab environment at the specified directory path. This allows seamless integration between the Colab notebook and the dataset which is stored in the Drive, which makes the data manipulation and data analysis directly from the cloud storage.

```
# Mounting Google-Drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

Mounted at /content/drive
```

Figure 4: Dataset Loading

6 Data Transformation

After installing the packages, video data is processed to extract pose landmarks from each frame and store the results in a Pandas DataFrame. MediaPipe's pose detection model identifies key landmarks, and the coordinates are organized into a structured DataFrame for detailed analysis and visualization of body movements.

```
def format_frame_landmarks(frame, detector):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_frame)
    landmark_coords = {}
    detection_result = detector.detect(mp_image)
    if detection_result.pose_landmarks:
        for pose_landmarks in detection_result.pose_landmarks:
                for idx, landmark in enumerate(pose_landmarks):
                    body_part = mp.solutions.pose.PoseLandmark(idx).name
                    landmark_coords[f'x_{body_part}'.lower()] = landmark.x
                    landmark_coords[f'y_{body_part}'.lower()] = landmark.y
                    landmark_coords[f'z_{body_part}'.lower()] = landmark.z

    return landmark_coords

def extract_pose_landmarks(path, input_type = "video"):
    base_options = python.BaseOptions(model_asset_path='pose_landmarker.task')
    options = vision.PoseLandmarkerOptions(
        base_options=base_options,
        output_segmentation_masks=True)

    detector = vision.PoseLandmarker.create_from_options(options)
```

Figure 5: Extract Pose Landmarks from Frames

At this stage the video files are processed to extract the frames and annotate them with the pose landmarks using the MediaPipe Library and using the process_video function reads the frames from a video and process each frame to detect the pose landmarks and saves the annotated frames as images in the output directory (video dataset/pose_landmark_images)

```python
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()
base_path = '/content/drive/MyDrive/video dataset/JumpingJack'
output_dir = '/content/drive/MyDrive/video dataset/pose_landmark_images/JumpingJack'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
def process_video(video_path, output_dir, video_name):
    cap = cv2.VideoCapture(video_path)
    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = pose.process(image_rgb)
        if results.pose_landmarks:
            mp.solutions.drawing_utils.draw_landmarks(
                frame, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
        frame_filename = os.path.join(output_dir, f'{video_name}_frame_{frame_count:04d}.png')
        cv2.imwrite(frame_filename, frame)
        frame_count += 1
    cap.release()
    return frame_count
video_files = [f for f in os.listdir(base_path) if f.endswith('.avi')]
total_frames_processed = 0
for video_file in video_files:
    video_path = os.path.join(base_path, video_file)
    video_name = os.path.splitext(video_file)[0]
    frames_processed = process_video(video_path, output_dir, video_name)
    total_frames_processed += frames_processed
    print(f'Processed {frames_processed} frames from video {video_file}.')
print(f'Total processed frames: {total_frames_processed} and saved to {output_dir}')
```

Figure 6: Pose Landmark Extraction for Videos

The below **Figure 7** has processed video files from specific exercise folders and extracts pose landmarks from each video using the **'extract_pose_landmarks'** function and then consolidates the results into a CSV file. This iterates through folders and video files and performs pose landmark extraction and saves the aggregated results to a CSV file for further process.

```python
base_path = '/content/drive/MyDrive/video dataset'
csv_file_path = '/content/drive/MyDrive/video dataset/Blazepose_custom_ds_final (4).csv'
 custom_pose_points_dataset  = '/content/drive/MyDrive/video dataset/Blazepose_custom_ds_final.csv'

for folder in os.listdir(base_path):
    master_df_list = []
    if folder in ['PushUps','JumpingJack', 'PullUps', 'BodyWeightSquats', 'BenchPress']:
      folder_path = os.path.join(base_path, folder)
      for files in os.listdir(folder_path):
          if files.endswith('.avi'):
              video_path = os.path.join(folder_path, files)
              print("Processing ", video_path)
              vid_frames_results_df = extract_pose_landmarks(video_path)
              if vid_frames_results_df is None:
                continue

              vid_frames_results_df['exercise'] = folder
              vid_frames_results_df['vid name'] = files
              master_df_list.append(vid_frames_results_df)

    else:
      continue
    combined_df = pd.concat(master_df_list, ignore_index=True)
    write_header = not os.path.isfile(csv_file_path)
    combined_df.to_csv(csv_file_path, mode='a', header=write_header, index=False)
```

Figure 7: Video Pose Data Extraction and Save to CSV

7 Exploratory Data Analysis

```
combined_df.shape
```

```
(50166, 101)
```

```
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50166 entries, 0 to 50165
Columns: 101 entries, x_nose to vid name
dtypes: float64(99), object(2)
memory usage: 38.7+ MB
```

```
# duplicate rows
duplicate_rows = combined_df[combined_df.duplicated()]
num_duplicates = len(duplicate_rows)
print(f'Number of duplicate rows: {num_duplicates}')
```

```
Number of duplicate rows: 0
```

Figure 8: Data information

The generated CSV file from the video folders for five strength training excercise consists of 101 features including exercise name and the video file name and 50166 rows.

```
print(combined_df.dtypes)
```

```
x_nose                float64
y_nose                float64
z_nose                float64
x_left_eye_inner      float64
y_left_eye_inner      float64
                       ...
x_right_foot_index    float64
y_right_foot_index    float64
z_right_foot_index    float64
exercise               object
vid name               object
Length: 101, dtype: object
```

```
print(combined_df.describe())
```

```
             x_nose        y_nose        z_nose  x_left_eye_inner  \
count  50166.000000  50166.000000  50166.000000      50166.000000
mean       0.499458      0.345463     -0.189184          0.502353
std        0.122187      0.187205      0.398930          0.125673
min        0.049007     -0.140063     -2.407700          0.035841
25%        0.445906      0.202554     -0.377798          0.449915
50%        0.501440      0.306598     -0.201130          0.504931
75%        0.549420      0.473567     -0.035501          0.550993
max        0.978234      0.956553      1.677908          0.984543
```

Figure 9: Data type and Data Description

```
# null values
null_values = combined_df.isnull().sum()
print("Null values in each column:")
print(null_values)

Null values in each column:
x_nose                0
y_nose                0
z_nose                0
x_left_eye_inner      0
y_left_eye_inner      0
                     ..
x_right_foot_index    0
y_right_foot_index    0
z_right_foot_index    0
exercise              0
vid name              0
Length: 101, dtype: int64
```

Figure 10: Checking for Null Values

8 Data Preprocessing

The figure below depicts the function **'calculate_required_files'** helps in identifying the additional files needed to reach a desired threshold for each folder. This function calculates 80% of the total files for each folder as training set. It then subtracts the number of files that have already been processed to find out how many more files are required for the testing set.

```
def calculate_required_files(total_files, processed_files=0, percentage_threshold=0.8):
    """
    Calculate the number of files needed to reach 80% for each folder.

    Parameters:
    total_files (dict): A dictionary with the total number of files for each folder.
    processed_files (int): The number of already processed files.

    Returns:
    dict: A dictionary with the remaining number of files needed to reach 80% for each folder.
    """
    required_files = {}
    for folder, total in total_files.items():
        eighty_percent = int(percentage_threshold* total)
        remaining = eighty_percent - processed_files
        required_files[folder] = max(remaining, 0)  # Ensure non-negative
    return required_files
```

Figure 11: Splitting Training and Testing data

The total number of files for each exercise class and the number of files required for training each class are as follows:

```
total files for each class: {'BodyWeightSquats': 112, 'BenchPress': 160, 'PullUps': 100, 'PushUps': 102, 'JumpingJack': 123}
{'BodyWeightSquats': 89, 'BenchPress': 128, 'PullUps': 80, 'PushUps': 81, 'JumpingJack': 98}
Processing exercise: weighted squats, Training files needed: 89
Processing exercise: bench press, Training files needed: 128
Processing exercise: pull ups, Training files needed: 80
Processing exercise: push ups, Training files needed: 81
Processing exercise: jumping jacks, Training files needed: 98
39985
10181
True
```

Figure 12: Exercise Class File Counts and Training Requirements

9 Model Training

Now, the classification model is trained using H2O library as shown below:

Training DRF, GBM and XGboost

```
# Model Training
from h2o.estimators import H2ORandomForestEstimator


drf_model = H2ORandomForestEstimator(
    ntrees=200,
    max_depth=70,
    balance_classes=True,
    seed=22224947
)


drf_model.train(x=x, y=y, training_frame=train)
```

Figure 13: Model Training using H2O

The above figure demonstrates the training of three different machine learning models using the H2O library. First, the necessary classes are imported, and each model is initialized with specific hyperparameters. The models are then trained on the provided dataset using the specified features and target column, allowing for a comparison of their performance.

```
DRF model saved to: /content/drive/MyDrive/video dataset/classification_model/DRF_model_python_1721988011695_1
GBM model saved to: /content/drive/MyDrive/video dataset/classification_model/GBM_model_python_1721988011695_2
XGBoost model saved to: /content/drive/MyDrive/video dataset/classification_model/XGBoost_model_python_1721988011695_3
```

Figure 14: Locations of Stored Trained Models

The trained models are saved in the specified directory, and their file paths are recorded in a JSON file named **“model_paths.json”** within the same directory.

10 Model Evaluation

At this stage evaluating the classification model by calculating accuracy, precision, recall, and F1 score and generates a classification report. The below figure11 includes a function to preprocess exercise names for consistency. These tools helps to assess and interpret the effectiveness of the model's predictions.

```python
#  Plot the confusion matrix heatmap
def plot_confusion_matrix(y_true, y_pred, class_names):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()

#  Evaluate the classification
def evaluate_classification(y_true, y_pred, class_names):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 Score: {f1:.2f}")

    print("\nClassification Report:")
    report = classification_report(y_true, y_pred, target_names=class_names, output_dict=True)
    print(classification_report(y_true, y_pred, target_names=class_names))

    plot_confusion_matrix(y_true, y_pred, class_names)

    return report
```

Figure 15: Classification Evaluation and Confusion Matrix

DRF Classfier Evaluation

```python
# Evaluating DRF
y_true = list(test_df['exercise'].apply(preprocess_pose_exercise_str))
y_pred = list(test_df['drf_prediction'])
class_names = set(y_true)
evaluate_classification(y_true, y_pred, class_names)
```

Figure 16: DRF Classfier Evaluation

Gradient Boosting Machine (GBM) Evaluation

```python
# Evaluating GBM
y_pred = list(test_df['gbm_prediction'])
evaluate_classification(y_true, y_pred, class_names)
```

Figure 17: Gradient Boosting Machine Model Implementation

Extreme Gradient Boosting (XGBoost) Evaluation

```
# Evaluating XGboost
y_pred = list(test_df['xgb_prediction'])
evaluate_classification(y_true, y_pred, class_names)
```

Figure 18: XgBoost Evaluation

The below figure 15 consolidates the prediction of each video by selecting the most frequent prediction from three models and it creates ' **result_df** ' Data frames map the most common predictions for each model.

```
drf_predicted_values = test_df.groupby('vid name')['drf_prediction'].agg(lambda x: x.value_counts().idxmax())
gbm_predicted_values = test_df.groupby('vid name')['gbm_prediction'].agg(lambda x: x.value_counts().idxmax())
xgb_predicted_values = test_df.groupby('vid name')['xgb_prediction'].agg(lambda x: x.value_counts().idxmax())

result_df = test_df[['exercise', 'vid name']].drop_duplicates().copy()
result_df['drf_prediction'] = result_df['vid name'].map(drf_predicted_values)
result_df['gbm_prediction'] = result_df['vid name'].map(gbm_predicted_values)
result_df['xgb_prediction'] = result_df['vid name'].map(xgb_predicted_values)
```

Figure 19: Consolidated Model Prediction

11 Model Prediction

Using pre-trained H2O model, a person's workout pose detection is detected by extract pose landmarks from the video which is given in base path and the extracted data converted into H2O frame

```
prediction = predict_exercise(gbm_model, input_type = "video", path = "")
print(f"\n\nWe have predicted that the person in the image is doing the following exercise: {prediction}\n")
# Measure runtime
start_time = time.time()
end_time = time.time()
runtime = end_time - start_time
print(f"Prediction took {runtime:.2f} seconds.")
```

```
Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please reru
Saving 5025960-hd_1080_1920_25fps.mp4 to 5025960-hd_1080_1920_25fps (2).mp4
Uploaded file: 5025960-hd_1080_1920_25fps (2).mp4
Can't receive frame (stream end?). Exiting ...
Parse progress: |████████████████████████████████████████| (done) 100%
gbm prediction progress: |████████████████████████████████| (done) 100%
Prediction counts for each label:
predict
jumping jacks      467
weighted squats    102
bench press          5
pull ups             2
Name: count, dtype: int64

The most common prediction across frames is: jumping jacks


We have predicted that the person in the image is doing the following exercise: jumping jacks
```

Figure 20: Testing Prediction

```python
from google.colab import files

def upload_vid_or_image():
    uploaded = files.upload()

    for filename in uploaded:
      content = uploaded[filename]
      with open(filename, 'wb') as f:
        f.write(content)

    if len(uploaded.keys()):
      FILE = next(iter(uploaded))
      print('Uploaded file:', FILE)
      return FILE
    else:
      return None

def preprocess_pose_exercise_str(exercise_str):
    exercise_map = {
        'Squats': 'weighted squats',
        'Pull': 'pull ups',
        'Push': 'push ups',
        'Jump': 'jumping jacks',
        'Bench': 'bench press'
    }

    for keyword, exercise in exercise_map.items():
        if keyword in exercise_str:
            return exercise
    return exercise_str

def predict_exercise(model, input_type = "image", path = ""):
  if not path:
    file_path = upload_vid_or_image()
  else:
    file_path = path

  df = extract_pose_landmarks(file_path, input_type = input_type)
  df_h2o_frame = h2o.H2OFrame(df)
  prediction = model.predict(df_h2o_frame).as_data_frame()
  final_prediction = prediction['predict'].value_counts().idxmax()

  return final_prediction
```

```python
prediction = predict_exercise(drf_model, input_type = "video", path = "")
print(f"\n\nWe have predicted that the person in the image is doing the following exercise: {prediction}\n")
```

Figure 21: Strength Training Exercise Prediction