Configuration Manual

MSc Research Project
MSc. Data Analytics

Anand Jha
Student ID: x21157251

School of Computing
National College of Ireland

Supervisor: Prof. Teerath kumar Menghwar

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Anand Jha
Student ID:	X21157251
Programme:	MSc. Data Analytics — **Year:** 2023-2024
Module:	MSc. Research Project
Lecturer:	Prof. Teerath Kumar Menghwar
Submission Due Date:	14/12/2023
Project Title:	Leveraging OpenCV for Precise Yoga Pose Estimation and Reducing Injury Risks
Word Count:	1271 — **Page Count:** 17

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: Anand Jha

Date: 14/12/2023

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

Anand Jha
x21157251

1 Introduction

This Configuration Manual outlines the comprehensive process involved in realizing the project “Leveraging OpenCV for Precise Yoga Pose Estimation and Reducing Injury Risks “. It details the specific aspects of data sources, system prerequisites, utilized libraries, and the code involved in the implementation and evaluation of the research models.

2 System Prerequisites

The hardware and software version for this project used are given below.

2.1 Hardware Prerequisites

Operating System	Windows 11
Processor	12th Gen Intel® core™ i9-12900Mhz, 14 core(s) 20 Logical Processor
Ram	16.0 GB
System type	X64-based PC

Table 1: Hardware Prerequisites

2.2 Software Prerequisites

Programming Language used in this project is Python and Jupiter Notebook is used as a programming tool to run the Python code.

- Python 3.9.13
- Jupiter Notebook 6.4.12

All the libraries used in the research to complete the project from start to end is shown in the Figure 1 and Figure 2.

```python
#!pip install mediapipe opencv-python pandas scikit-learn
import mediapipe as mp #Meidapipe
import cv2 #opencv
import csv
import numpy as np
import os
import sys
import tqdm
import random
import pandas as pd
import pickle # to save a model

# Package for visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Packages for model Implementation and Evaluation
from sklearn.metrics import confusion_matrix,classification_report
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
```

Figure 1: Python Libraries used in this Python Project- Part 1

```python
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import SpatialDropout1D
from keras.layers import InputLayer
from keras.layers import Conv1D
from keras.layers import Flatten
from keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
from keras.layers import MaxPooling1D
```

Figure 2: Python Libraries used in this Python Project- Part 2

3 Data Collection

This project draws upon a curated yoga image dataset sourced from Kaggle, consisting of diverse yoga poses to classify and detect in real-time. The dataset encompasses five prevalent yoga poses: Asho Mukha Scanasana (320 images), Balasana (261 images), Utkata Konasana (180 images), Virabhadrasana (209 images), and Vrikshasana (334 images). In total, 1304 yoga pose images were meticulously compiled, featuring individuals with varying backgrounds and body types. These poses were systematically organized into distinct folders, each named after the corresponding yoga pose, facilitating structured data management and analysis.

4 Data Preprocessing

In order to transform the yoga image dataset into a CSV file format containing 99 landmark features (33 landmark points * 3 dimensions), several crucial steps were undertaken in research project code. Before proceeding with the data preprocessing phase, it was imperative to install essential libraries such as Mediapipe and OpenCV. Additionally, all the requisite packages, as outlined in Figure 1 and Figure 2 of the project documentation, were imported to facilitate the dataset conversion process. These preparatory steps ensured the seamless execution of the subsequent data preprocessing procedures.

4.1 Feature Extraction

In this part all the images are extracted from the yoga pose folder name and with the help of Mediapipe library all the 33 landmarks in three different direction is extracted. This landmark is then stored in the csv file format.

So, there will be in total 99 features in this csv file format. Refer Figure 3and Figure 4.

```
#Initialising input image path and  output csv path
images_in_folder = 'C:\\Users\\jhaan\\Downloads\\Dataset'
#images_out_folder = 'fitness_poses_images_out_basic'
#csv_out_path = 'yoga_poses_landmark_dataset.csv'

#from mediapipe.solutions import drawing_utils as mp_drawing
mp_drawing = mp.solutions.drawing_utils # Drawing helpers
# from mediapipe.solutions import pose as mp_pose
mp_pose= mp.solutions.pose # Blazepose pose estimation model
```

Figure 3: Initialize Path and Load Blazepose model

```
import os
import csv
import sys
import cv2
import tqdm
import numpy as np
import mediapipe as mp

# Set the paths for input images and the output CSV file

csv_out_path = 'yoga_poses_landmark_dataset1.csv'
images_out_folder = 'fitness_poses_images_out_basic'

# Iterating through each Folder, converting images into landmark points and saving them to a CSV file
with open(csv_out_path, 'w') as csv_out_file:
    csv_out_writer = csv.writer(csv_out_file, delimiter=',', quoting=csv.QUOTE_MINIMAL)

    # folder names are used as pose class names
    pose_class_names = sorted([n for n in os.listdir(images_in_folder) if not n.startswith('.')])
    counter = 1
    for pose_class_name in pose_class_names:
        print('Extracting landmark points from dataset', pose_class_name, file=sys.stderr)
        if not os.path.exists(os.path.join(images_out_folder, pose_class_name)):
            os.makedirs(os.path.join(images_out_folder, pose_class_name))
        image_names = sorted([
            n for n in os.listdir(os.path.join(images_in_folder, pose_class_name))
            if not n.startswith('.')
        ])
        for image_name in tqdm.tqdm(image_names, position=0):
            # Load image.
            input_frame = cv2.imread(os.path.join(images_in_folder, pose_class_name, image_name))
            input_frame = cv2.cvtColor(input_frame, cv2.COLOR_BGR2RGB)

           # Applying Blazepose model on the image and extracting 33 3D landmark points.
            with mp_pose.Pose() as pose_tracker:
                result = pose_tracker.process(image=input_frame)
                pose_landmarks = result.pose_landmarks

            # Save landmark points to a csv file.
            if pose_landmarks is not None:

                # check the number of landmarks and take pose landmarks.
                assert len(pose_landmarks.landmark) == 33, 'Unexpected number of predicted pose landmarks: {}'.format(len(pose_landmarks.landmark))
                pose_landmark = [[lmk.x, lmk.y, lmk.z] for lmk in pose_landmarks.landmark]

                # Write pose sample to CSV.
                pose_landmarks = np.around(pose_landmark, 5).flatten().astype(str).tolist()
                csv_out_writer.writerow([pose_class_name] + pose_landmarks)
```

Figure 4: Creating CSV file for landmarks in x,y,z direction

Heading to each column is give as x1, y1, z1, x2, y2, z2 and so on. Refer Figure 5.

```
df_csv = pd.read_csv('./yoga_poses_landmark_dataset1.csv', header=None)

l1=['pose_name']
for i in range(1,34):
    l1.append('x'+str(i))
    l1.append('y'+str(i))
    l1.append('z'+str(i))

df_csv.to_csv('./yoga_poses_landmark_dataset_new11.csv', header=l1, index=False)

df_csv1 = pd.read_csv('./yoga_poses_landmark_dataset_new11.csv')
df_csv1
```

Figure 5: Adding Header to the Dataset

5 Data Transformation

As all the data in sequence order of yoga below code is used to shuffle the dataset, so that there will be no bias in the model. Refer Figure 6.

```
import csv
import random

# Set the path for your CSV file
csv_file_path = './yoga_poses_landmark_dataset_new11.csv'

# Read the CSV file into a list
with open(csv_file_path, 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    rows = list(csv_reader)


# Separate the header (first row) and the data rows
header = rows[0]
data = rows[1:]

# Shuffle the rows randomly
random.shuffle(data)

# Write the shuffled rows back to the CSV file, including the header
with open(csv_file_path, 'w', newline='') as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(header)  # Write the header
    csv_writer.writerows(data)  # Write the shuffled data rows

print("CSV file rows shuffled (excluding header) successfully.")

CSV file rows shuffled (excluding header) successfully.

df = pd.read_csv('./yoga_poses_landmark_dataset_new11.csv')
df
```

Figure 6: Shuffling the Dataset

To visualize the distribution of the class Following code is used below, refer to the Figure 7. And Later Label encoding is done in Figure 8.

```
import matplotlib.pyplot as plt

# Assuming 'df' contains your dataset and 'pose_name' is the column representing classes
class_counts = df['pose_name'].value_counts()

# Plotting a bar plot for class distribution
plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color='skyblue')
plt.title('Class Distribution')
plt.xlabel('Yoga Poses')
plt.ylabel('Number of Instances')
plt.xticks(rotation=45)  # Rotating x-labels for better readability
plt.tight_layout()
plt.show()
```

Figure 7: Class Distribution Graph

Feature Encoding

```
# Encode the response variable into numrical values
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['label_enc'] = labelencoder.fit_transform(df['pose_name'])
```

```
classes= df[['label_enc','pose_name']].drop_duplicates()
classes
```

Figure 8: Feature Encoding

After Feature encoding, to add the pose name to encoded classes use the below code, refer Figure 9.

```
# Adding pose names and encode values to a dictionary for display purposes
classes.set_index('label_enc', inplace= True)
yoga_pose=classes.to_dict()
yoga_pose_dict=yoga_pose['pose_name']
yoga_pose_dict
```

Figure 9: Labelling the Class

Before Training the model, dataset must split into training and test in 70:30. Refer Figure 10

Splitting data into train test split - 70-30 ratio

```
X=df.drop(['pose_name','label_enc'], axis=1) # independent variable
y=df['label_enc'] # dependent variable
```

```
A=X.columns
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3, random_state=1234,stratify=y)
```

Figure 10: Splitting the data

6 Model Implementation and Evaluation

Both deep learning and Machine learning algorithm are used to train the model, to evaluate deep learning Figure 11 and 15) mode accuracy and loss function is used, and graph is drawn for the same (Figure 12,13 and 14). In the case of machine learning evaluation matrix like accuracy, precision, recall and f2 score is used to find the best model. And model is further optimising by using the Hyper parameter.

6.1 Deep learning

LSTM Model Implementation

```
# reshaping the input array before applying to lstm
y_train_re= to_categorical(y_train).astype(int)
y_test_re= to_categorical(y_test).astype(int)

X_train_lstm= np.array(X_train)
X_test_lstm = np.array(X_test)
X_train_lstm=X_train_lstm.reshape(X_train_lstm.shape[0],1,X_train_lstm.shape[1])
X_test_lstm=X_test_lstm.reshape(X_test_lstm.shape[0],1,X_test_lstm.shape[1])
```

```
#LSTM Model Implementation
#tf.set_random_seed(122)
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', dropout=0.2, input_shape=(1,X_train_lstm.shape[2])))

model.add(LSTM(128, return_sequences=True, dropout=0.2, activation='relu'))

model.add(LSTM(64, return_sequences=False, dropout=0.2, activation='relu'))

model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(5, activation='softmax'))

#Compile
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
#Fitting LSTM model on train data
history_model_lstm=model.fit(X_train_lstm, y_train_re, epochs=200, validation_data=(X_test_lstm,y_test_re))
```

Figure 11: LSTM model

```
model.evaluate(X_test_lstm,y_test_re)

11/11 [==============================] - 0s 13ms/step - loss: 0.4182 - categorical_accuracy: 0.9020
[0.4181867241859436, 0.9020172953605652]

#Plotting Accuracy and Loss Curve
%matplotlib inline
import matplotlib.pyplot as plt
acc= history_model_lstm.history['categorical_accuracy']
val_acc = history_model_lstm.history['val_categorical_accuracy']
loss = history_model_lstm.history['loss']
val_loss = history_model_lstm.history['val_loss']


epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Testing accuracy')
plt.title('Training and Testing accuracy - LSTM')
plt.legend()
plt.figure()

plt.plot(epochs, loss,'r', label='Training Loss')
plt.plot(epochs,val_loss,'b', label='Testing Loss')
plt.title('Training and Testing loss - LSTM')
plt.legend()

plt.show()
```

Figure 12: Accuracy and Loss Graph

```
# Confusion report
from sklearn.metrics import confusion_matrix,classification_report
y_predicted = model.predict(X_test_lstm)
y_pred=[]
for i in y_predicted:
    y_pred.append(np.argmax(i))
y_pred1=pd.Series(y_pred)
y_pred1
print(classification_report(y_test,y_pred1))
```

Figure 13: Confusion Report For LSTM

```
# Confusion metric
from sklearn import metrics
import seaborn as sns
cm=metrics.confusion_matrix(y_test,y_pred1)
# crete seaborn heatmap with required labels
sns.heatmap(cm,annot =True,cmap='Reds', fmt='g',xticklabels=yoga_pose_dict.values(), yticklabels=yoga_pose_dict.values())
```

Figure 14: Confusion Matrix for LSTM

To implement and evaluate CNN use below Figure 15.

1D CNN Model Implementation

```
# Reshaping the input array before applying to 1D CNN
X_train_re=np.array(X_train)
X_test_re=np.array(X_test)
sample_size=X_train_re.shape[0]
time_steps=X_test_re.shape[1]
input_dim=1
X_train_re=X_train_re.reshape(sample_size,time_steps,input_dim)
X_train_re.shape
sample_size1=X_test_re.shape[0]
time_steps1=X_test_re.shape[1]
input_dim1=1
X_test_re=X_test_re.reshape(sample_size1,time_steps1,input_dim1)
X_test_re.shape
```

```
(347, 99, 1)
```

```
# 1D CNN model Implementation

model_cnn = Sequential()
model_cnn.add(Conv1D(128,kernel_size=3,input_shape=(X_train_re.shape[1],1)))
model_cnn.add(Dropout(0.5))
model_cnn.add(MaxPooling1D(pool_size=1,name='MaxPooling1D'))
model_cnn.add(Flatten())
model_cnn.add(Dropout(0.5))
model_cnn.add(Dense(64, activation='relu'))
model_cnn.add(Dense(8, activation='relu'))
model_cnn.add(Dense(5, activation='softmax'))
# Compiling the model
model_cnn.compile(optimizer='Adam',loss='categorical_crossentropy', metrics=['categorical_accuracy'])
# Fitting the model on train data
history_model_cnn=model_cnn.fit(X_train_re, y_train_re, epochs=200,validation_data=(X_test_re,y_test_re))
```

Figure 15: CNN Model

```
#Evaluate the 1DCNN model on test data
model_cnn.evaluate(X_test_re,y_test_re)
```

```
11/11 [==============================] - 0s 20ms/step - loss: 0.3825 - categorical_accuracy: 0.9135
[0.3825025260448456, 0.9135446548461914]
```

```
# Visualize Loss and Accuracy Plot of the 1D CNN
%matplotlib inline
import matplotlib.pyplot as plt
acc= history_model_cnn.history['categorical_accuracy']
val_acc = history_model_cnn.history['val_categorical_accuracy']
loss = history_model_cnn.history['loss']
val_loss = history_model_cnn.history['val_loss']


epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Testing accuracy')
plt.title('Training and Testing accuracy - 1D CNN')
plt.legend()
plt.figure()

plt.plot(epochs, loss,'r', label='Training Loss')
plt.plot(epochs,val_loss,'b', label='Testing Loss')
plt.title('Training and Testing loss - 1D CNN')
plt.legend()

plt.show()
```

Figure 16: CNN Model Evaluation Graph

6.2 Machine learning

To Implement Machine learning model, refer below codes. Figure 17 is implementation for Random Forest Model.

Applying Machine Learning Models on GeneratedLand_Mark dataset

```
# Fitting Generated Landmark datset on Random Forest classifier
import numpy as np
seed = np.random.seed(22)
rng=np.random.RandomState(3)
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

#RandomizedSearchCV for hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
params= {'n_estimators':[10,20,30,40,50,60,70,80,90,100], 'max_features': ['log2','sqrt'],'max_depth':[2,4,6,8,10],'min_samples_split':[2,5],'min_samples_leaf':[1,2],'bootstrap':[True,False]}
random_forest=RandomizedSearchCV(RandomForestClassifier(random_state=rng),param_distributions=params,n_iter=5,scoring='accuracy',n_jobs=-1,cv=cv,verbose=3,random_state=rng)
random_forest.fit(X_train,y_train)
```

Figure 17: Random Forest Model

Refer Figure 18, 19 for the evaluation matrix.

```
# Random Forest model Evaluation
# best parameters
print(random_forest.best_params_)

{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 10, 'bootstrap': False}

print("Accuracy is:", random_forest.score(X_test,y_test))

Accuracy is: 0.9077809798270894

from sklearn import metrics
import seaborn as sns
y_pred_random=random_forest.predict(X_test)
random_forest_cm = metrics.confusion_matrix(y_test,y_pred_random)
# create seaborn heatmap with required labels
sns.heatmap(random_forest_cm,annot = True,cmap='Reds',fmt='g',xticklabels=yoga_pose_dict.values(),yticklabels=yoga_pose_dict.values())
```

Figure 18: Evaluation Matrix for Random Forest Model- Part 1

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy-Random forest:",round((metrics.accuracy_score(y_test, y_pred_random))*100,2))
print("Precision-Random Forest:",round((metrics.precision_score(y_test, y_pred_random, average="macro"))*100,2))
print("Recall-Random Forest:", round((metrics.recall_score(y_test, y_pred_random, average="macro"))*100,2))
print("F1 Score -RandomForest:",round((metrics.f1_score(y_test,y_pred_random,average="macro"))*100,2))
```

Figure 19: Evaluation Matrix for Random Forest Model- Part 2

To implement Xgboost Classifier refer Figure 20,21,22.

```
### XgBoost Classifier
seed = np.random.seed(22)
rng = np.random.RandomState(2)
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from xgboost import XGBClassifier

# RandomizedSearchCV for Hyperparameter tuning
params={'alpha':[0.001,0.01,0.1],'max_depth':[1,2,3,4,5,10], 'learning_rate':[0.1,0.25,0.5]}
xgboost=RandomizedSearchCV(XGBClassifier(random_state=rng),param_distributions=params,n_iter=5,scoring='accuracy',n_jobs=-1,cv=cv,verbose=3,random_state=rng)
xgboost.fit(X_train,y_train)
```

Figure 20: XGBoost Classifier Model

```
print(xgboost.best_params_)
print("Accuracy is:", xgboost.score(X_test,y_test))

{'max_depth': 3, 'learning_rate': 0.25, 'alpha': 0.1}
Accuracy is: 0.9164265129682997

from sklearn import metrics
import seaborn as sns
y_pred_xgboost=xgboost.predict(X_test)
cm = metrics.confusion_matrix(y_test,y_pred_xgboost)
# create seaborn heatmap with required labels
sns.heatmap(cm,annot = True,cmap='Reds',fmt='g',xticklabels=yoga_pose_dict.values(),yticklabels=yoga_pose_dict.values())
```

Figure 21: XGBoost Classifier Evaluation Matrix Part 1

```
print("Accuracy-xgboost:",round((metrics.accuracy_score(y_test, y_pred_xgboost))*100,2))
print("Precision-xgboost:",round((metrics.precision_score(y_test, y_pred_xgboost, average="macro"))*100,2))
print("Recall-xgboost:", round((metrics.recall_score(y_test, y_pred_xgboost, average="macro"))*100,2))
print("F1 Score -xgboost:",round((metrics.f1_score(y_test,y_pred_xgboost,average="macro"))*100,2))
```

Figure 22: XGBoost Classifier Evaluation Matrix Part 2

To implement SVM refer Figure 22,23,24.

```
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
seed=np.random.seed(33)
rng=np.random.RandomState(3)
cv= RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

param_grid={'C':[0.1,1,10,20,50,100],'kernel':['rbf','poly','sigmoid','linear'],'degree':[1,2,3,4,5,6]}
model_svm=RandomizedSearchCV(SVC(random_state =rng),param_distributions=param_grid,n_iter=5,scoring='accuracy',n_jobs=-1,cv=cv,verbose=3,random_state =rng)
model_svm.fit(X_train,y_train)
```

Figure 23: SVM Model

```
print(model_svm.best_params_)
print("Accuracy is:", model_svm.score(X_test,y_test))

{'kernel': 'poly', 'degree': 6, 'C': 1}
Accuracy is: 0.9077809798270894

from sklearn import metrics
import seaborn as sns
y_pred_svm=model_svm.predict(X_test)
svm_cm = metrics.confusion_matrix(y_test,y_pred_svm)
# create seaborn heatmap with required labels
sns.heatmap(svm_cm,annot = True,cmap='Reds',fmt='g',xticklabels=yoga_pose_dict.values(),yticklabels=yoga_pose_dict.values())
```

Figure 24: SVM Model Evaluation Matrix Part 1

```
print(classification_report(y_test,y_pred_svm))

              precision    recall  f1-score   support

           0       0.99      0.97      0.98        88
           1       0.92      0.92      0.92        65
           2       0.83      0.83      0.83        52
           3       0.96      0.81      0.88        59
           4       0.84      0.95      0.89        83

    accuracy                           0.91       347
   macro avg       0.91      0.90      0.90       347
weighted avg       0.91      0.91      0.91       347

print("Accuracy-svm:",round((metrics.accuracy_score(y_test, y_pred_svm))*100,2))
print("Precision-svm:",round((metrics.precision_score(y_test, y_pred_svm, average="macro"))*100,2))
print("Recall-svm:", round((metrics.recall_score(y_test, y_pred_svm, average="macro"))*100,2))
print("F1 Score -svm:",round((metrics.f1_score(y_test,y_pred_svm,average="macro"))*100,2))
```

Figure 25: SVM Model Evaluation Matrix Part 2

To implement decision model, refer following Figure 26,27,28.

```
### Decision tree Classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
seed = np.random.seed(44)
rng= np.random.RandomState(4)
cv= RepeatedStratifiedKFold(n_splits=5,n_repeats=3, random_state=1)
param_grid={'max_depth':[10,30,50,60,90,100],'max_features':['auto','sqrt','log2'],'min_samples_split':[2,4,6]}
model_decision= RandomizedSearchCV(DecisionTreeClassifier(random_state =rng),param_distributions=param_grid,n_iter=5,scoring='accuracy',n_jobs=-1,cv=cv,verbose=3,random_state=rng)
model_decision.fit(X_train,y_train)
```

Figure 26: Decision Tree Classifier

```
print(model_decision.best_params_)
print("Accuracy is:", model_decision.score(X_test,y_test))

{'min_samples_split': 2, 'max_features': 'sqrt', 'max_depth': 30}
Accuracy is: 0.8645533141210374

from sklearn import metrics
import seaborn as sns
y_pred_decision=model_decision.predict(X_test)
decision_cm = metrics.confusion_matrix(y_test,y_pred_decision)
# create seaborn heatmap with required labels
sns.heatmap(decision_cm,annot = True,cmap='Reds',fmt='g',xticklabels=yoga_pose_dict.values(),yticklabels=yoga_pose_dict.values())
```

Figure 27: Decision Tree Classifier Evaluation Matrix Part 1

```
print("Accuracy-decision tree:",round((metrics.accuracy_score(y_test, y_pred_decision))*100,2))
print("Precision-decision tree:",round((metrics.precision_score(y_test, y_pred_decision, average="macro"))*100,2))
print("Recall-decision tree:", round((metrics.recall_score(y_test, y_pred_decision, average="macro"))*100,2))
print("F1 Score -decision tree:",round((metrics.f1_score(y_test,y_pred_decision,average="macro"))*100,2))
```

Figure 28: Decision Tree Classifier Evaluation Matrix Part2

To Implement KNN Model, refer Figure 29,30.

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

k_values = list(range(1, 20))
accuracy_values = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)

plt.plot(k_values, accuracy_values, marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. k Values')
plt.show()
```

Figure 29: KNN Model

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Figure 30: KNN Evaluation Matrix

7 Saving the Optimal Model

Evaluate and save the model with highest accuracy and dump it in pickle file and load it when the system is built for future use.

saving the model

```
import pickle
file_name='xgb-reg_fix2.pkl'
pickle.dump(xgboost,open(file_name,"wb"))
xgb_model_loaded=pickle.load(open("xgb-reg_fix.pkl","rb"))
```

```
xgb_model_loaded=pickle.load(open("xgb-reg_fix.pkl","rb"))
```

Figure 31: Saving the Model

8 Building Yoga Pose Image Detection System

Refer Below Figure 32 to build the system to identify pose in the given image .

```
def poseclassify(path):
    sample_img=cv2.imread(path)
    plt.figure(figsize=[10,10])
    plt.title("Sample_image")
    plt.axis("off")
    plt.imshow(sample_img)
    plt.imshow(sample_img[:,:,::-1])
    plt.show()

    # Define pose estimation and skeltal image drawing object
    mp_drawing = mp.solutions.drawing_utils
    mp_pose = mp.solutions.pose
    pose = mp_pose.Pose(static_image_mode=True,model_complexity=2)
    img_copy = sample_img.copy()
    #Convert BGR to RGB format and apply the Blazepose pose estimation model
    results = pose.process(cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB))


    # to draw it in 3D Space
    mp_drawing.plot_landmarks(results.pose_world_landmarks,mp_pose.POSE_CONNECTIONS)

    # Extract the landmark data and pass to xgboost model for prediction
    landmarks = results.pose_landmarks.landmark

    # Flatten Array
    pose_row = list(np.array([[landmark.x, landmark.y,landmark.z] for landmark in landmarks]).flatten())
    X = pd.DataFrame([pose_row])

    # Xgboost prediction
    X.columns = A
    body_language_class = xgb_model_loaded.predict(X)[0]
    body_language_prob = xgb_model_loaded.predict_proba(X)[0]
    print(body_language_class, body_language_prob)

    pose_detected= yoga_pose_dict[body_language_class]
    prob=(round(body_language_prob[np.argmax(body_language_prob)],2))
    print(" The pose detected is:", pose_detected)
    print("probablity :", prob)

    if results.pose_landmarks:
        mp_drawing.draw_landmarks(img_copy,results.pose_landmarks,mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=((255,127,80)), thickness=1,circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(50,205,50),thickness =1,circle_radius=2)
                                  )
        fig=plt.figure(figsize=[10,10])
        plt.title("output-image")
        plt.axis("off")
        plt.imshow(img_copy[:,:,::-1])
        plt.show()
```

Figure 32: Yoga Pose Image Detection

To call the system use the code in Figure 33.

```
poseclassify("C:\\Users\\jhaan\\Desktop\\Anand_01.jpg")
```

Figure 33: Identify the Yoga Pose

9 Building the Real Time Yoga Pose Detector

To build the final real time yoga pose detector refer the Figure 34, 35, 36. In this Figure 34 has code to add yoga pose benefit and Figure 35 has Audio feedback Function.

```python
def yogabenifit(yoga_pose):
    if yoga_pose == "VIRABHADRASANA":
        cv2.putText(image,"It strengthen the arms, shoulders, and legs.", (20, frame.shape[0] - 100),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"It maintain balance in the body.", (20, frame.shape[0] - 80),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"It increase stamina, muscle endurance and relieve tension", (20, frame.shape[0] - 60),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)

    if yoga_pose == "Adho Mukha Svanasana":
        cv2.putText(image,"It may help improve digestion.", (20, frame.shape[0] - 100),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"It might help stimulate circulation.", (20, frame.shape[0] - 80),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"It may help to relieve leg pain and ankle pain.", (20, frame.shape[0] - 60),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)

    if yoga_pose == "UTKATA KONASANA":
        cv2.putText(image," It strengthens the pelvic floor, thighs, knees, and ankles.", (20, frame.shape[0] - 100),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"It strengthens the spine and activates the Muladhara, Swadhisthan, and Manipur chakra. ", (20, frame.shape[0] - 80),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"It also activates the heart chakra and promotes confidence and compassion.", (20, frame.shape[0] - 60),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)


    if yoga_pose == "BALASANA":
        cv2.putText(image,"Enhances blood circulation", (20, frame.shape[0] - 100),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"Strengthens the ligaments in the knees", (20, frame.shape[0] - 80),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"Effectively calms the mind", (20, frame.shape[0] - 60),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)


    if yoga_pose == "VRIKSHASANA":
        cv2.putText(image,"Boosts the balance of the body", (20, frame.shape[0] - 100),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"Improves the posture.", (20, frame.shape[0] - 80),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
        cv2.putText(image,"Tones the muscles of your legs.", (20, frame.shape[0] - 60),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,0,0),1,cv2.LINE_AA)
```

Figure 34: Building Yoga Pose Benefit Function

```python
#pip install pyttsx3
import pyttsx3

# Initialize the text-to-speech engine
engine = pyttsx3.init()
def text2speech(yoga_pose):

    engine.say(f"it is {yoga_pose} pose ")
    engine.runAndWait()
```

Figure 35: Adding Audio Feature In The Sytem

```python
cap = cv2.VideoCapture(0)

with mp_pose.Pose(min_detection_confidence=0.9, min_tracking_confidence=0.9) as pose:
    while cap.isOpened():

        ret, frame =cap.read()

        image= cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        results = pose.process(image)

        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(255,127,80), thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(50,205,50), thickness=2, circle_radius=2))



        try:
            if results.pose_landmarks is not None:

                landmarks = results.pose_landmarks.landmark
                pose_row=list(np.array([[landmark.x,landmark.y,landmark.x] for landmark in landmarks]).flatten())
                X= pd.DataFrame([pose_row])
                X.columns = A
                body_language_class = xgb_model_loaded.predict(X)[0]
                body_language_prob =xgb_model_loaded.predict_proba(X)[0]

                yoga_pose= yoga_pose_dict[body_language_class]
                #print("pose detected:", yoga_pose)
                prob=(round(body_language_prob[np.argmax(body_language_prob)],2))

                if prob>=0.85:
                    grade="Very Good"

                if prob<0.85 and prob>=0.80:
                    grade="Good"

                if prob<0.80:
                    grade="Needs Improvement"

                cv2.rectangle(image,(0,0), (1080,60), (0,0,16), -1)

                if prob>=0.75 and len(landmarks) >= 25:
```

Figure 36: Yoga Pose Detector System- Part 1

```python
    if prob>=0.75 and len(landmarks) >= 25:
        cv2.putText(image,"Yoga Pose Detected",(150,12), cv2.FONT_HERSHEY_SIMPLEX,0.5,(255,255,255), 1,cv2.LINE_AA)
        cv2.putText(image, yoga_pose,(150,35),cv2.FONT_HERSHEY_SIMPLEX,0.5,(255,255,255), 1, cv2.LINE_AA)

        if prob>=0.80:

            yogabenifit(yoga_pose)


        cv2.putText(image,"Probability", (30,12),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),1,cv2.LINE_AA)
        cv2.putText(image,str(round(body_language_prob[np.argmax(body_language_prob)],2)), (30,35),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),1,cv2.LINE_AA)

        cv2.putText(image,"System_comment", (400,12),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),1,cv2.LINE_AA)

        cv2.putText(image,grade, (400,35),cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),1,cv2.LINE_AA)

    else:

        cv2.putText(image,"No pose detected", (180,30),cv2.FONT_HERSHEY_SIMPLEX, 0.8,(255,255,255),1,cv2.LINE_AA)
        cv2.putText(image,"please let full body come into the frame", (100,50),cv2.FONT_HERSHEY_SIMPLEX, 0.8,(255,255,255),1,cv2.LINE_AA)


    if prob >= 0.75:
        if not in_pose:
            in_pose = True
            pose_start_time = cap.get(cv2.CAP_PROP_POS_MSEC)
            # Speak when the pose is initially detected


        else:
            pose_end_time = cap.get(cv2.CAP_PROP_POS_MSEC)
            total_time = (pose_end_time - pose_start_time) / 1000  # Convert to seconds


            # Display the result
            cv2.putText(image, f"Total time in {yoga_pose}: {total_time} seconds",
                        (150, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 1, cv2.LINE_AA)



        if not in_pose:
            in_pose = True
            pose_start_time = cap.get(cv2.CAP_PROP_POS_MSEC)

        else:
```

```python
                    else:
                        pose_end_time = cap.get(cv2.CAP_PROP_POS_MSEC)
                        total_time = (pose_end_time - pose_start_time) / 1000


                        if total_time >= 2:  # Example: Speak after 5 seconds in the pose
                            # Speak after certain time spent in the pose
                            text2speech(yoga_pose)



                else:
                    in_pose = False

            #else:

                #print("")

        except Exception as e:
            print(e)



        cv2.namedWindow("Resized_pose_detection_window", cv2.WINDOW_NORMAL)

        cv2.resizeWindow("Resized_pose_detection_window",2000, 1000)

        cv2.imshow("Resized_pose_detection_window", image)


        if cv2.waitKey(10) & 0xFF == ord('q'):
            break



cap.release()
cv2.destroyAllWindows()
```

Figure 37: Yoga Pose Detector System- Part 2

References

Link for the dataset used in this Project.