Real-Time Yoga Pose Detection using Machine Learning Algorithm

Research Project - Configuration Manual
MSc.Data Analytics

Jothika Sunney
Student ID: x20224532

School of Computing
National College of Ireland

Supervisor: Paul Stynes, Pramod Pathak, Musfira Jilani

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Jothika Sunney
Student ID:	x20224532
Programme:	MSc.Data Analytics
Year:	2022
Module:	Research Project - Configuration Manual
Supervisor:	Paul Stynes, Pramod Pathak, Musfira Jilani
Submission Due Date:	15/08/2022
Project Title:	Real-Time Yoga Pose Detection using Machine Learning Algorithm
Word Count:	1473
Page Count:	14

Signature:	Jothika Sunney
Date:	16th September 2022

Real-Time Yoga Pose Detection using Machine Learning Algorithm

Jothika Sunney
x20224532

1 Introduction

This Configuration manual describes the steps involved in implementing the research project 'Real Time Yoga Pose Detection Using Machine Learning Algorithm'. Specific details of the data sources, system specifications, libraries, and code used for implementation and evaluation of the research models are disclosed in the manual.

Section 2 presents the System requirements for the research. Section 3 describes the Data Collection. Section 4 defines the steps involved in Data preprocessing. A description of the data transformation is provided in Section 5. The sixth section describes the steps involved in the implementation and evaluation of different models. The following section 7 describes the detection of yoga pose from an image, the section 8 presents the implementation steps for real-Time yoga pose detection and the Final section concludes the report.

2 System Requirements

The Hardware and Software Requirements for project Implementation is explained here

2.1 Hardware Requirements

Operating System	Windows 10
Processor	AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz
Installed RAM	8.00 GB
System Type	64-bit operating system, x64-based processor

Table 1: Hardware Requirements

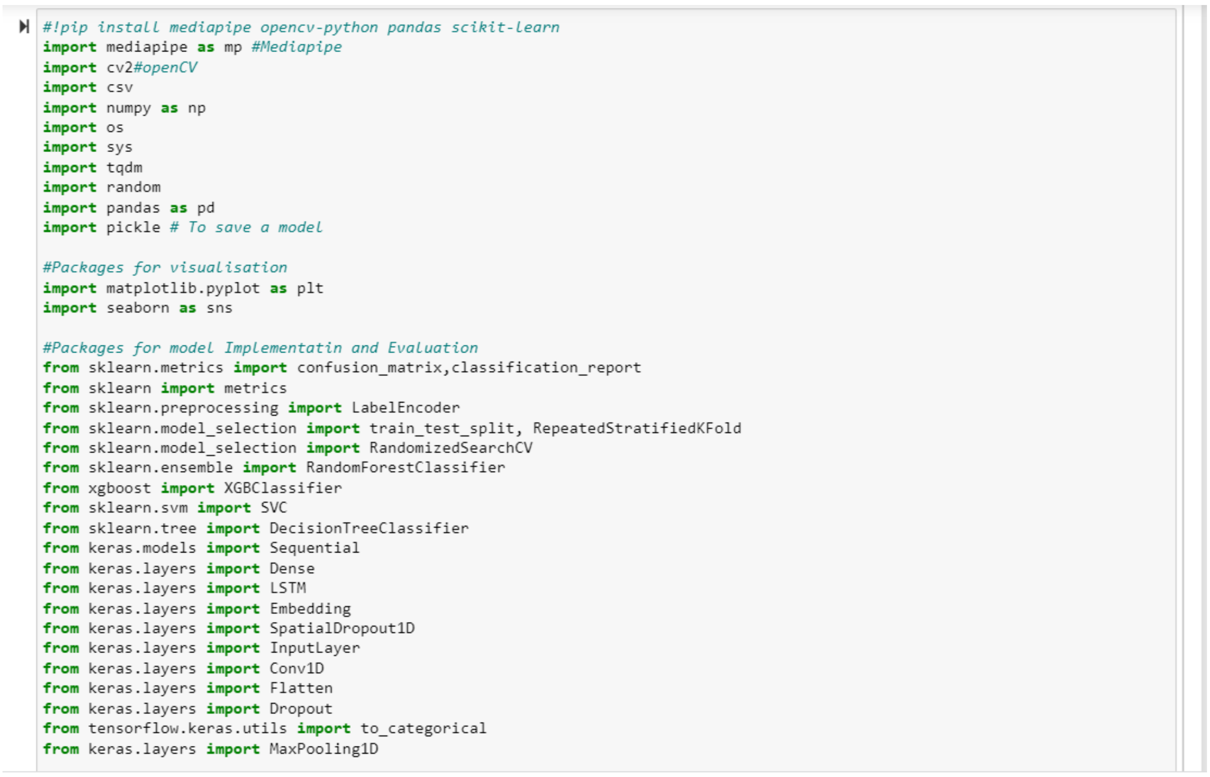
2.2 Software Requirements

Python programming language was used for project implementation. The python code was written and executed in Jupyter notebook.

- Python 3.8.8

- Jupyter Notebook
- Anaconda Navigator

Figure.1 shows libraries which were used in this research project for Data preprocessing, Data Transformation, Evaluation and Implementation of Machine Learning and Deep learning models and for implementing real-time Yoga pose detection Framework.



```python
#!pip install mediapipe opencv-python pandas scikit-learn
import mediapipe as mp #Mediapipe
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

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

#Packages for model Implementatin 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
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 1: All Required Python libraries for the Research Project

3 Data Collection

This project uses a publicly available yoga image dataset from Kaggle to classify 5 common yoga poses.[1]. The poses examined in this study were downdog (320 images), goddess (260 images), plank (381 images), tree (229 images), and warrior (361 images). A total of 1551 yoga pose images were included in this collection, consists of different individuals with diverse background. Yoga poses were organized into folders with names corresponding to the respective yoga pose.

4 Data Preprocessing

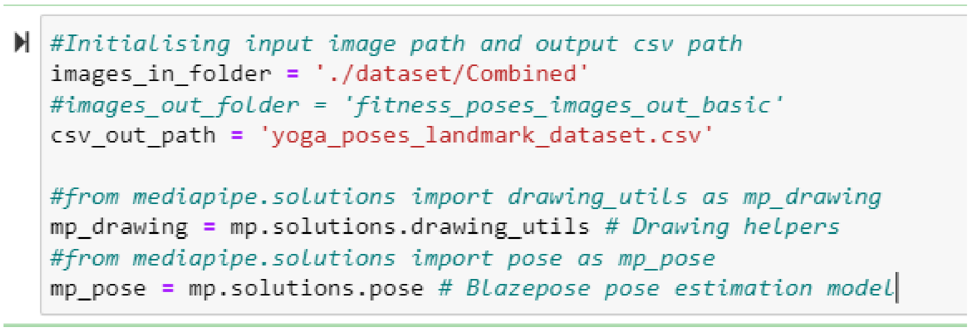
All the steps involved in converting the yoga image dataset into a csv file with 99 (33 Landmark points*3 Dimensions) landmark features and response variable are discussed

[1]Yoga Pose Image Dataset :`https://www.kaggle.com/datasets/niharika41298/yoga-poses-dataset`

in this section. Install Mediapipe and OpenCV before the Data preprocessing step, and import all the necessary packages mentioned in Figure.1.

4.1 Feature Extraction

Following the definition of the input image path and output csv file, the Blazepose model and Drawing helpers were imported from the Mediapipe library. Blazepose model was used for 3D Landmark detection and Drawing helpers were used to draw the skeleton image. Refer Figure.2 [2].



```
#Initialising input image path and output csv path
images_in_folder = './dataset/Combined'
#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 2: Initialize path and Load Blazepose model

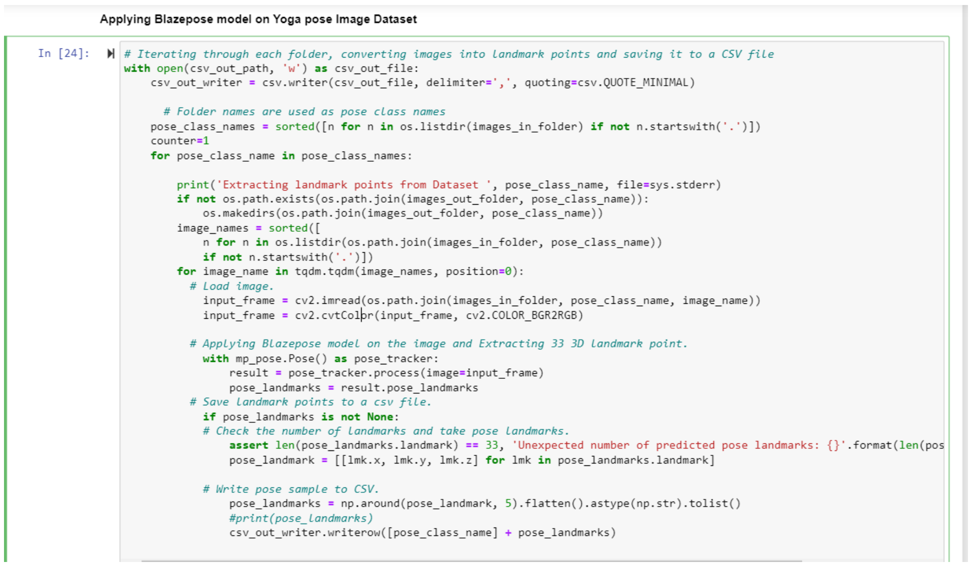
In the next step. The images were read from individual folders, then applied to the Blazepose model to detect skeltal images. After that, the 33 3D landmark points extracted from the skeltal image were appended to a CSV file. The Blazepose model detects landmark arrays as four-dimensional arrays with directions x,y,z and visibility component. The visibility component has been excluded from the features. So a total of 33 * 3 , ie 99 features were generated by the model for each image. Since the features were in standarized form , additional scaling was not required. Each landmark point was then appended to a CSV file along with the corresponding folder name indicating the yoga pose. Lastly, a header was added to the csv file with the pose name and the names of the landmark points x, y, and z. Refer Figure.3,4 for the code snippet.

5 Data Transformation

Yoga pose name is the response variable which is a multiclass categorical variable. It needs to be label encoded into numerical values before applying to any model. Refer Figure.5 for the Feature Encoding steps. The pose names and encoded values were added to a dictionary for display purposes.

After Label Encoding, the dataset was seperated into predictor and response variable and then splitted the data into 70:30 ratio for training and testing respectively. Refer Figure.6

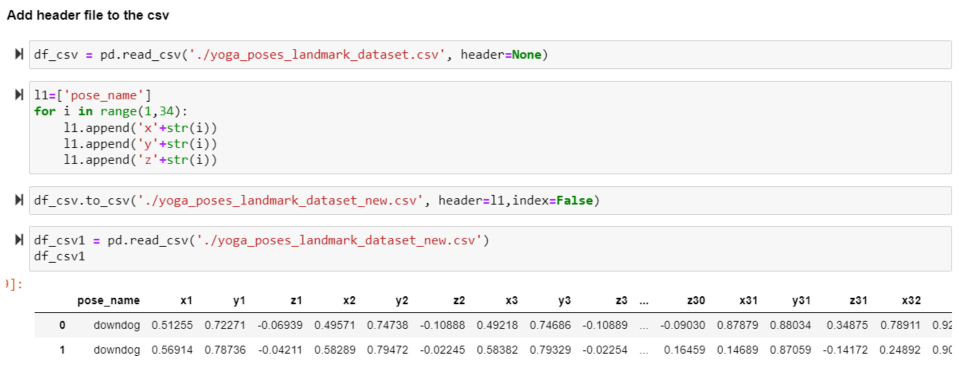
[2]Mediapipe Blazepose :https://google.github.io/mediapipe/solutions/pose.html



Applying Blazepose model on Yoga pose Image Dataset

```
In [24]:  # Iterating through each folder, converting images into landmark points and saving it 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 point.
                      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(pos
                          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(np.str).tolist()
                          #print(pose_landmarks)
                          csv_out_writer.writerow([pose_class_name] + pose_landmarks)
```

Figure 3: Converting Image Dataset into 33 3D landmark points and appending to a CSV



Add header file to the csv

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

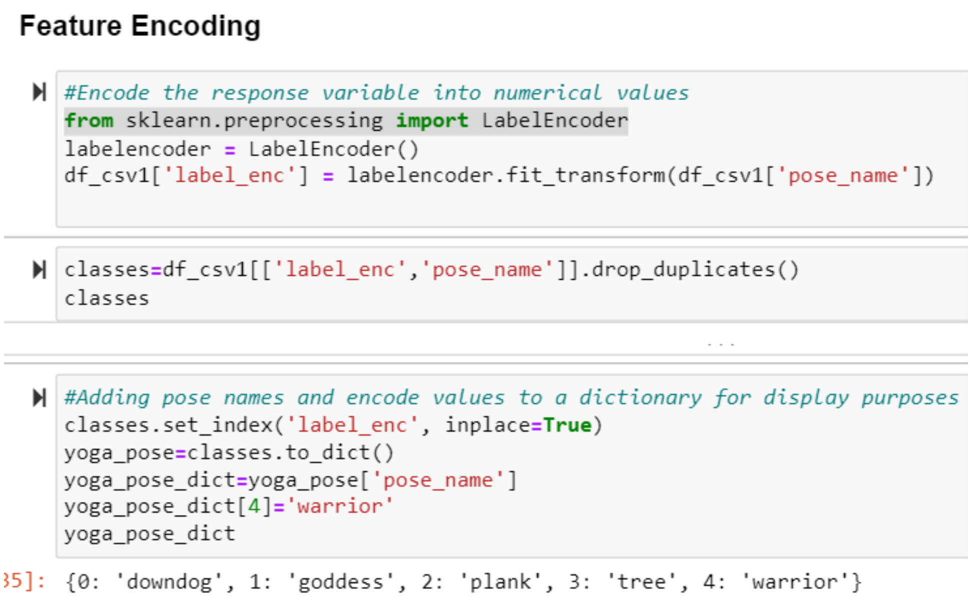
```
l1=['pose_name']
for i in range(1,34):
    l1.append('x'+str(i))
    l1.append('y'+str(i))
    l1.append('z'+str(i))
```

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

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

```
]:
     pose_name       x1       y1        z1       x2       y2        z2       x3       y3        z3  ...      z30      x31      y31       z31      x32
  0    downdog  0.51255  0.72271  -0.06939  0.49571  0.74738  -0.10888  0.49218  0.74686  -0.10889  ...  -0.09030  0.87879  0.88034   0.34875  0.78911  0.92
  1    downdog  0.56914  0.78736  -0.04211  0.58289  0.79472  -0.02245  0.58382  0.79329  -0.02254  ...   0.16459  0.14689  0.87059  -0.14172  0.24892  0.9
```

Figure 4: Adding header to the CSV file



Feature Encoding

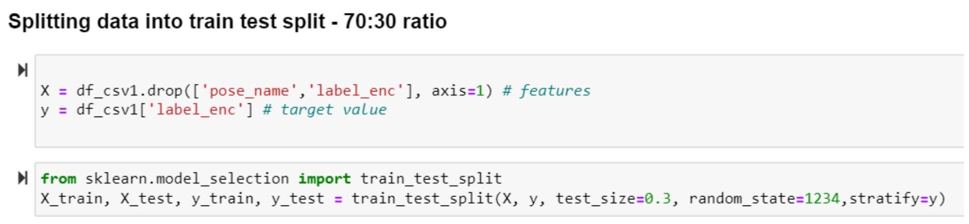
```
#Encode the response variable into numerical values
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df_csv1['label_enc'] = labelencoder.fit_transform(df_csv1['pose_name'])
```

```
classes=df_csv1[['label_enc','pose_name']].drop_duplicates()
classes
```

```
#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[4]='warrior'
yoga_pose_dict
```

```
35]: {0: 'downdog', 1: 'goddess', 2: 'plank', 3: 'tree', 4: 'warrior'}
```

Figure 5: Label Encoding response variable



Splitting data into train test split - 70:30 ratio

```
X = df_csv1.drop(['pose_name','label_enc'], axis=1) # features
y = df_csv1['label_enc'] # target value
```

```
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 6: Train Test split 70:30 ratio

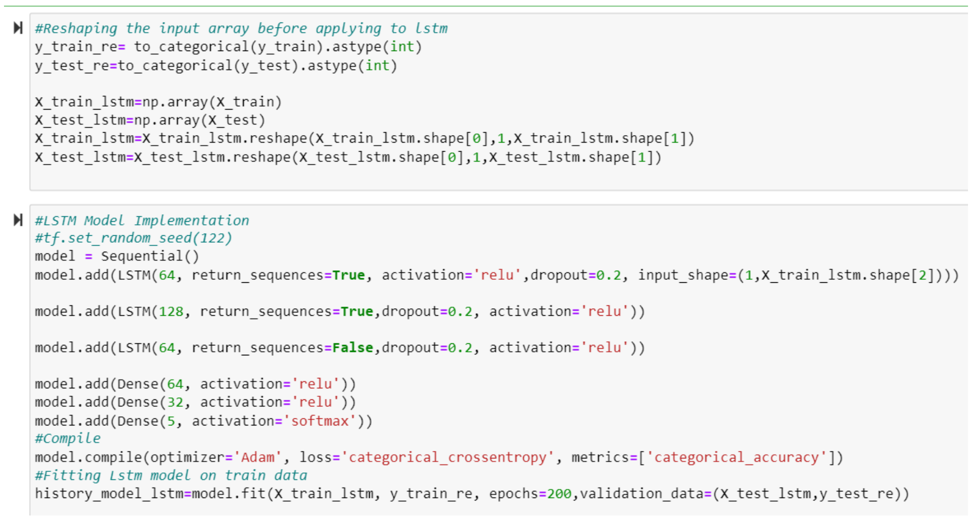
6 Model Implementation and Evaluation

As part of the project, two deep learning models - LSTM, 1D CNN and four machine learning models - Random Forest, XgBoost, SVM classifier and Decision Tree were Implemented and evaluated. The Models were analysed based on accuracy, precision, Recall ,F1 Score ,Time Complexity and Model Complexity.

6.1 Applying LSTM Model on generated landmark dataset

6.1.1 LSTM model Implementation

Before applying the LSTM model all the necessary libraries were imported, then the landmark dataset was reshaped into an array of size (sample_Size,1,99) where 99 is the total number of features. Figure.7 shows the the code screenshot for 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 7: LSTM Model Implementation

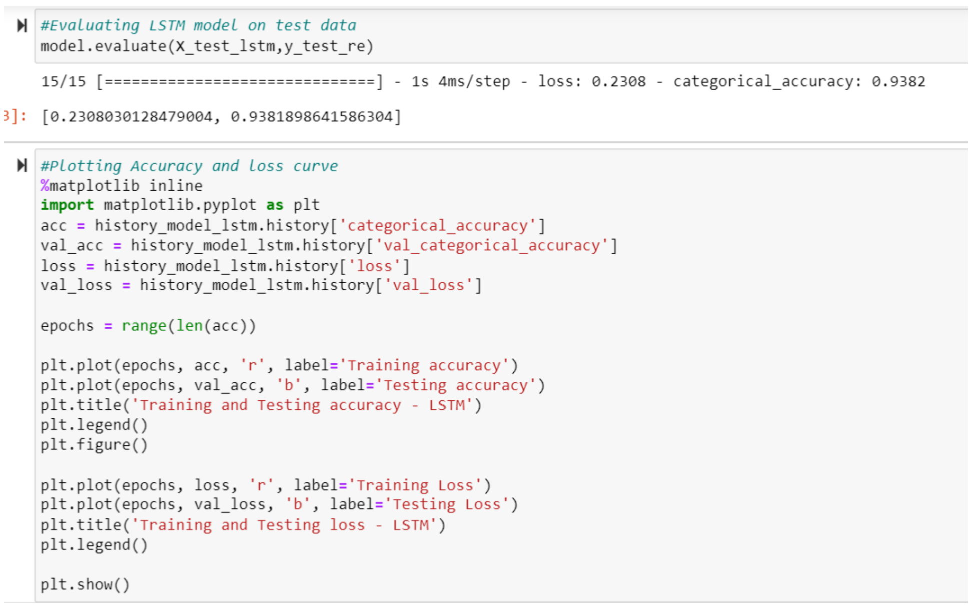
6.1.2 LSTM model Evaluation

LSTM model was evaluated on the test dataset. Accuracy and loss plot were analysed for model performance(Refer Figure.8). Matplotlib.pyplot and seaborn were the main visualisation libraries used. The confusion metrics and confusion report were evaluated to identify the model's performance for each yoga pose. Code for confusion metric and confusion report was replicated for other models as well. Refer Figure.9 for the code snippet.

6.2 Applying 1D CNN Model on generated landmark dataset

6.2.1 1D CNN Model Implementation

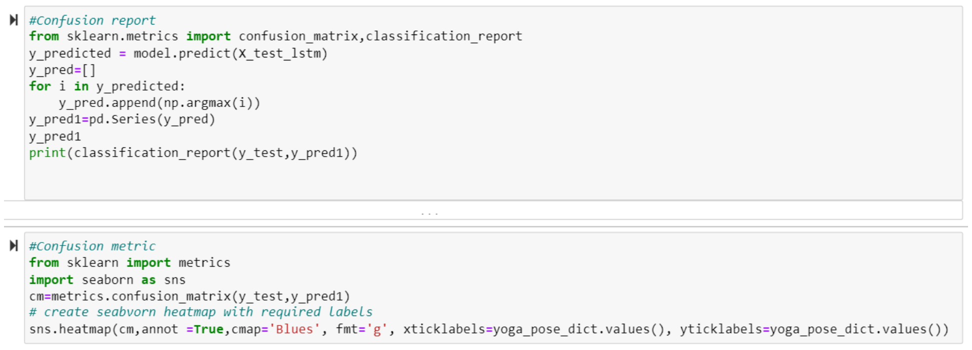
The landmark data was reshaped into an array of size (sample_Size,99,1) and inputted to the 1D CNN .Figure.10 shows the the code screenshot for CNN model Implementation.



```
#Evaluating LSTM model on test data
model.evaluate(X_test_lstm,y_test_re)

15/15 [==============================] - 1s 4ms/step - loss: 0.2308 - categorical_accuracy: 0.9382

3]: [0.2308030128479004, 0.9381898641586304]
```

```
#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()
```

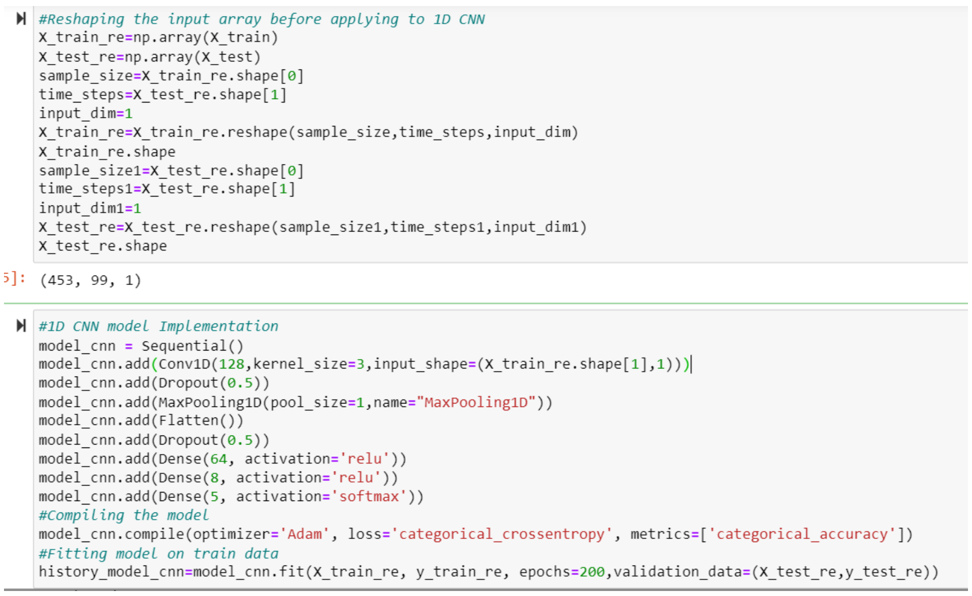
Figure 8: LSTM Model Evaluation on Test data and Accuracy/Loss plot visualization



```
#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))
```

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

Figure 9: LSTM confusion report and confusion plot (same code has been replicated for other models.



```
#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
```

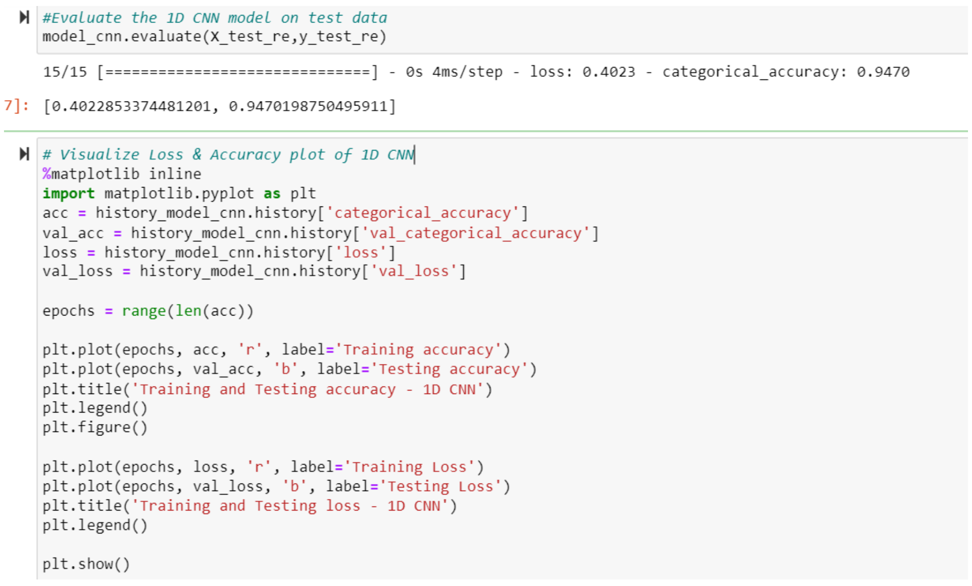
```
5]: (453, 99, 1)
```

```
#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 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 10: 1D CNN Model Implementation

6.2.2 1D CNN Model Evaluation

The confusion metrics and report code was replicated for CNN based on Figure.9 mentioned in previous section. Code snippet of Accuracy/Loss plot and Evaluation on test data for 1D CNN is shown in the Figure.11



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

```
15/15 [==============================] - 0s 4ms/step - loss: 0.4023 - categorical_accuracy: 0.9470
```

```
7]: [0.4022853374481201, 0.9470198750495911]
```

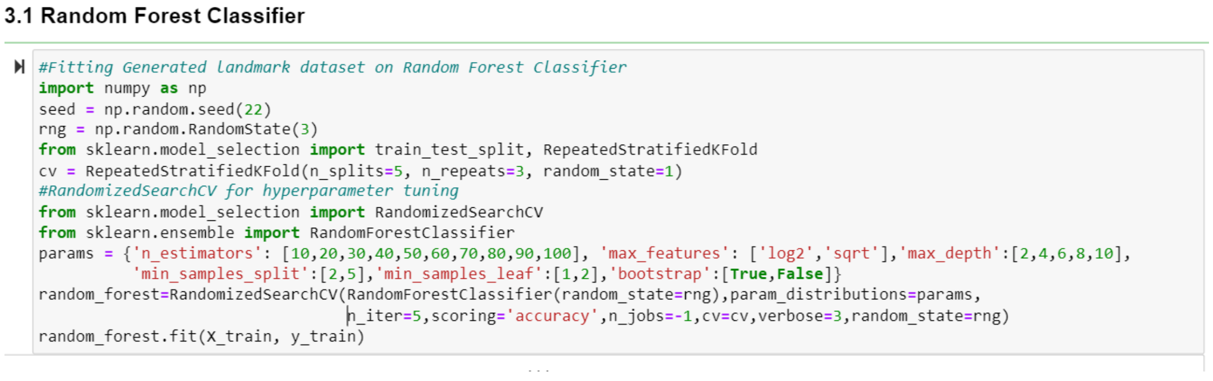
```
# Visualize Loss & Accuracy plot of 1D CNN
%matplotlib inline
import matplotlib.pyplot as plt
acc = history_model_cnn.history['categorical_accuracy']
val_acc = history_model_cnn.history['val_categorical_accuracy']
loss = history_model_cnn.history['loss']
val_loss = history_model_cnn.history['val_loss']

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 11: 1D CNN Model Evaluation

6.3 Applying Machine Learning Models on Generated Landmark Dataset

Four Different Classification models - Random Forest, XgBoost ,Support Vector Classifier ,Decision Tree Classifier were evaluated on the preprocessed 3D Landmark dataset genertaed from Blazepose model. This section depicts code for implementation and evalauation of all the four Machine Learning Classifiers.

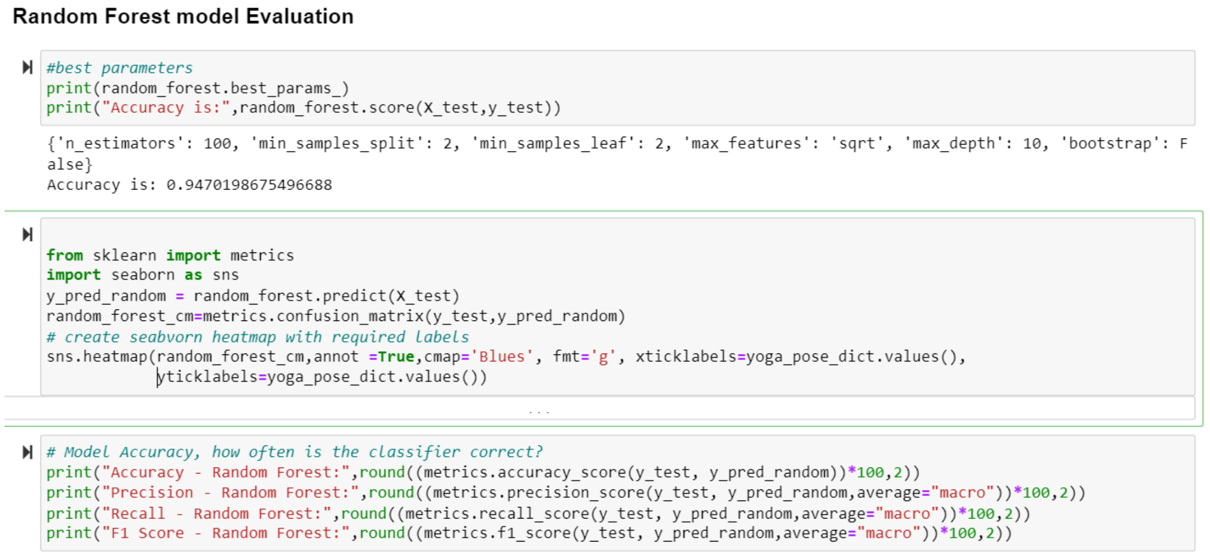
Refer Figure.12 for Random Forest Implementation and Figure.13 for Random Forest Evaluation. Figure.14,15 shows the code for XgBoost Classifier Implementation and Evaluation respectively and Figure.16, Figure.17 shows the code for Support vector Implementation and Evaluation. Similarly Figure.18, Figure.19 shows the code for Decision Tree Implementation and Evaluation respectively.



3.1 Random Forest Classifier

```
#Fitting Generated landmark dataset 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 12: Random Forest Implementation



Random Forest model Evaluation

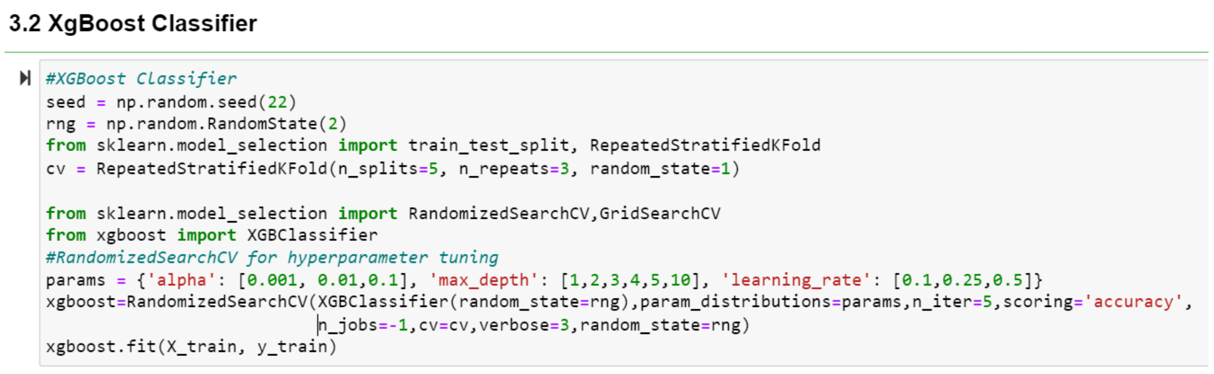
```
#best parameters
print(random_forest.best_params_)
print("Accuracy is:",random_forest.score(X_test,y_test))
```

```
{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 10, 'bootstrap': F
alse}
Accuracy is: 0.9470198675496688
```

```
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 seabvorn heatmap with required labels
sns.heatmap(random_forest_cm,annot =True,cmap='Blues', fmt='g', xticklabels=yoga_pose_dict.values(),
           yticklabels=yoga_pose_dict.values())
```

```
# 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 - Random Forest:",round((metrics.f1_score(y_test, y_pred_random,average="macro"))*100,2))
```

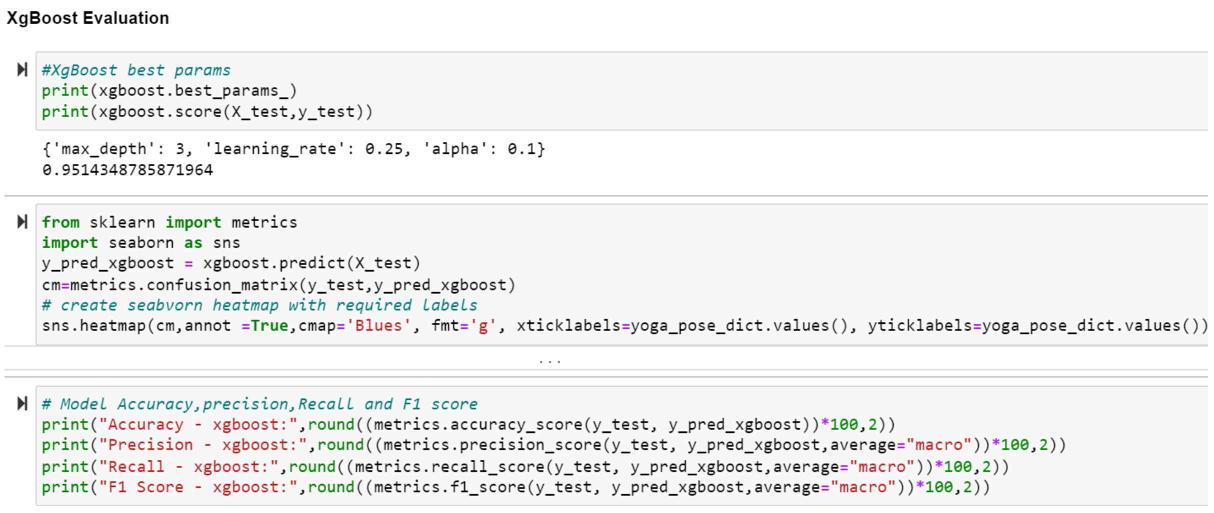
Figure 13: Random Forest Evaluation



3.2 XgBoost Classifier

```
#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 14: XgBoost Implementation



XgBoost Evaluation

```
#XgBoost best params
print(xgboost.best_params_)
print(xgboost.score(X_test,y_test))
```

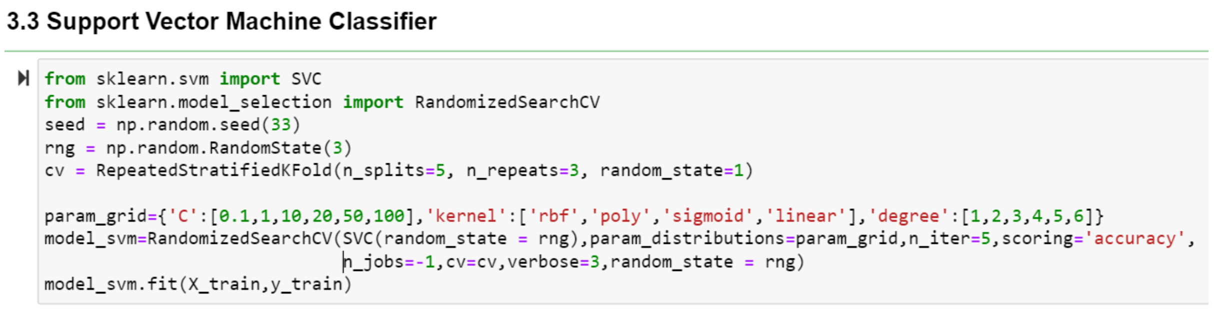
```
{'max_depth': 3, 'learning_rate': 0.25, 'alpha': 0.1}
0.9514348785871964
```

```
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 seabovrn heatmap with required labels
sns.heatmap(cm,annot =True,cmap='Blues', fmt='g', xticklabels=yoga_pose_dict.values(), yticklabels=yoga_pose_dict.values())
```

...

```
# Model Accuracy,precision,Recall and F1 score
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))
```

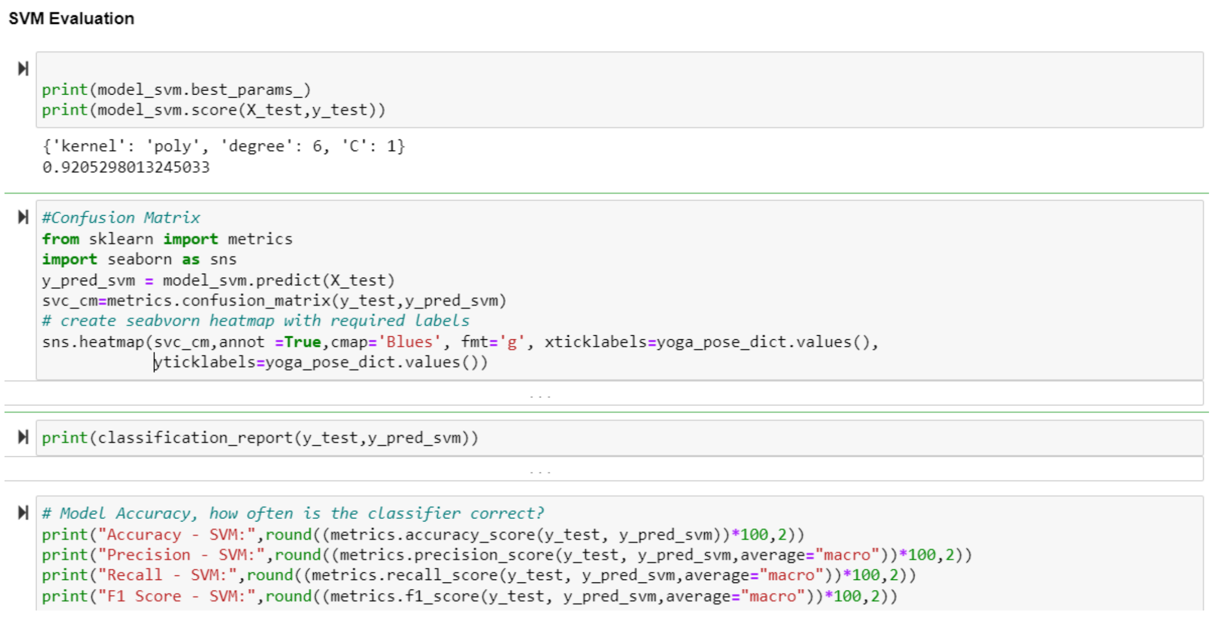
Figure 15: XgBoost Evaluation



3.3 Support Vector Machine Classifier

```
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 16: Support vector classifier Implementation



SVM Evaluation

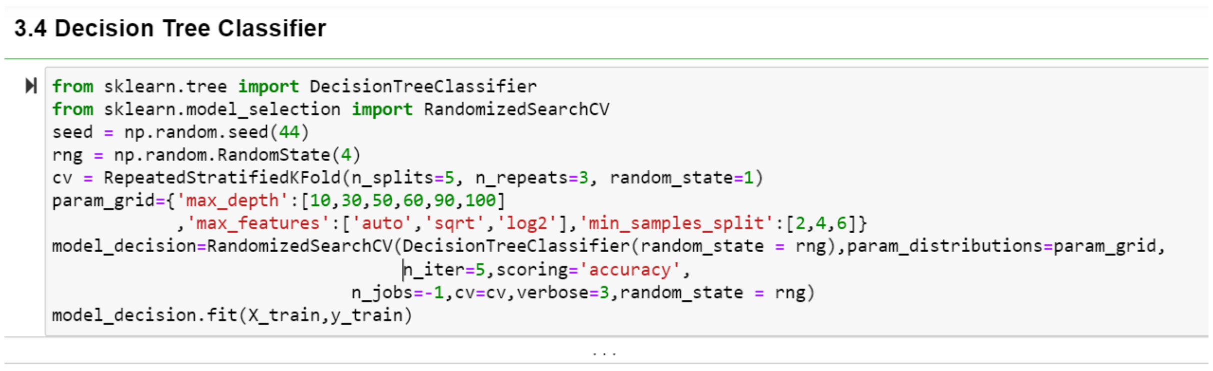
```
print(model_svm.best_params_)
print(model_svm.score(X_test,y_test))

{'kernel': 'poly', 'degree': 6, 'C': 1}
0.9205298013245033
```

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

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

```
# Model Accuracy, how often is the classifier correct?
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))
```

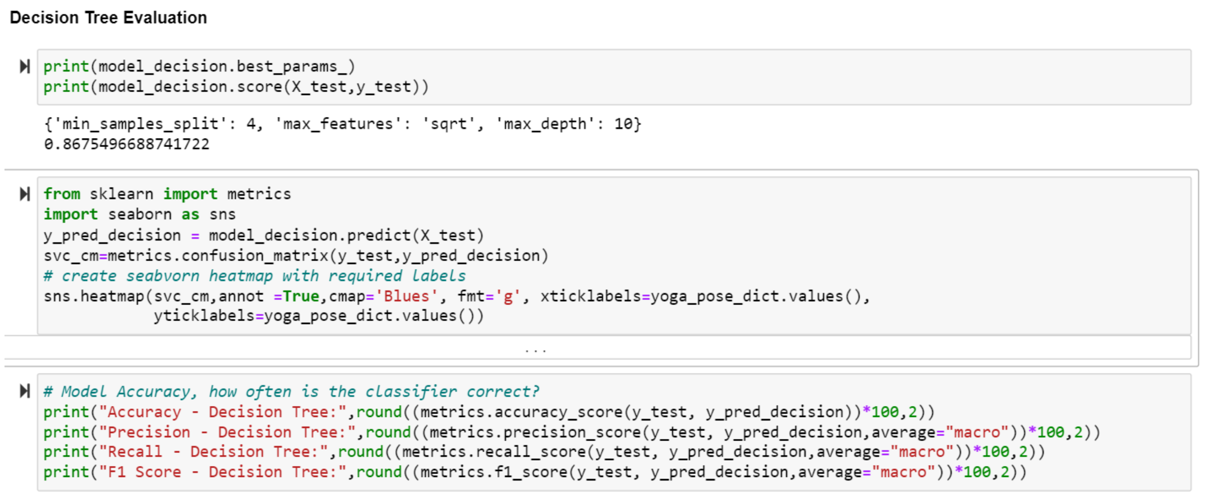
Figure 17: Support vector classifier Evaluation



3.4 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 18: Decision Tree Implementation



Decision Tree Evaluation

```
print(model_decision.best_params_)
print(model_decision.score(X_test,y_test))

{'min_samples_split': 4, 'max_features': 'sqrt', 'max_depth': 10}
0.8675496688741722
```

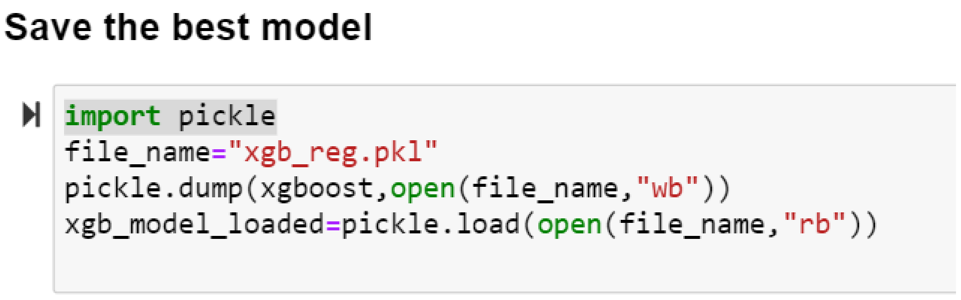
```
from sklearn import metrics
import seaborn as sns
y_pred_decision = model_decision.predict(X_test)
svc_cm=metrics.confusion_matrix(y_test,y_pred_decision)
# create seaborn heatmap with required labels
sns.heatmap(svc_cm,annot =True,cmap='Blues', fmt='g', xticklabels=yoga_pose_dict.values(),
    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

```
# Model Accuracy, how often is the classifier correct?
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 19: Decision Tree Evaluation

6.4 Save the optimum model

Based on the performance metrics analysis and model complexity, XgBoost is chosen as the optimum classifier to be used with the Blazepose model for real-Time Yoga Pose Prediction. Pickle library is used to store and load the trained Xgboost classifier as "Xgb_model_loaded". Ref Fig.20 for the code snippet.


Save the best model

```
import pickle
file_name="xgb_reg.pkl"
pickle.dump(xgboost,open(file_name,"wb"))
xgb_model_loaded=pickle.load(open(file_name,"rb"))
```

Figure 20: Save XgBoost Classification Model

7 Predicting Yoga pose from an Image

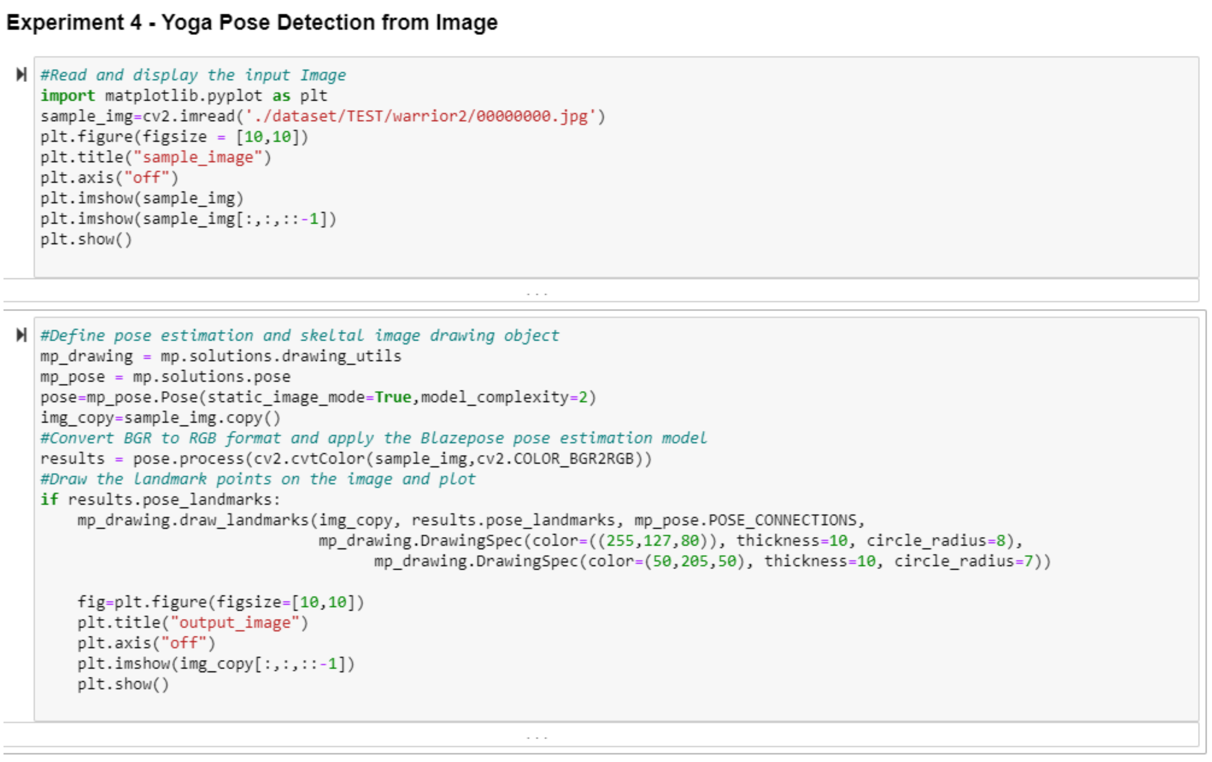
The proposed framework was tested on a yoga pose sample image before being applied to real-time video. First, a warrior pose image from the test dataset was loaded, and then the Blazepose model was applied to extract the 3D landmark points. After the landmark points were flattened, they were converted into a dataframe and passed to the XGBoost classifier for prediction. The model correctly predicts the yoga pose as "warrior". The implementation steps can be found in Figure.21 and Figure.22.

8 Real-Time Yoga Pose Detection Framework

This project provides a cost-effective solution for detecting yoga poses in real-time. Blazepose model, XgBoost Classifier, and Computer Vision methodologies were combined to develop the Framework. A real-time video was captured from webcam using the Videocapture object of the openCV, features were extracted using the Blazepose model, and then predicted using the XgBoost classifier. Users were provided with real-time feedback on yoga poses, probability, and grades. Refer Figure.23, Figure.24 and Figure.25 for Implementation steps. The prerequisites for real-time yoga pose detection is that the user should be within 2 to 3 metres of the webcam, with the whole body visible.

9 Conclusion

The configuration manual explains the complete implementation of the research step by step. Each section of the report has been explained in detail and sequentially to help the reader replicate the process.

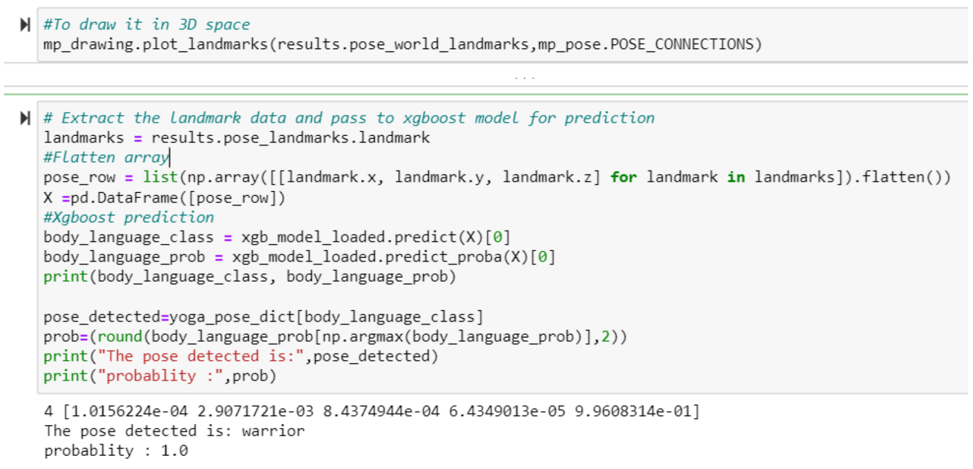


Experiment 4 - Yoga Pose Detection from Image

```
#Read and display the input Image
import matplotlib.pyplot as plt
sample_img=cv2.imread('./dataset/TEST/warrior2/00000000.jpg')
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))
#Draw the landmark points on the image and plot
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=10, circle_radius=8),
                                    mp_drawing.DrawingSpec(color=(50,205,50), thickness=10, circle_radius=7))

    fig=plt.figure(figsize=[10,10])
    plt.title("output_image")
    plt.axis("off")
    plt.imshow(img_copy[:,:,::-1])
    plt.show()
```

Figure 21: Yoga pose Image Detection - Part 1



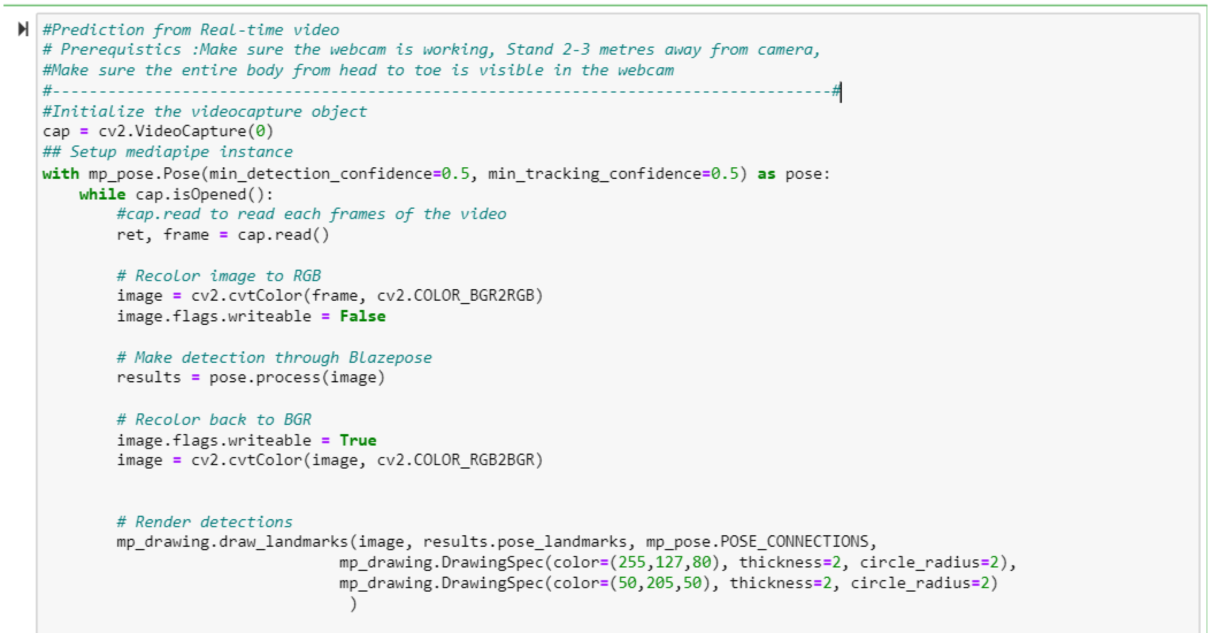
```
#To draw it in 3D space
mp_drawing.plot_landmarks(results.pose_world_landmarks,mp_pose.POSE_CONNECTIONS)
```

```
# 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
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)
```

```
4 [1.0156224e-04 2.9071721e-03 8.4374944e-04 6.4349013e-05 9.9608314e-01]
The pose detected is: warrior
probablity : 1.0
```

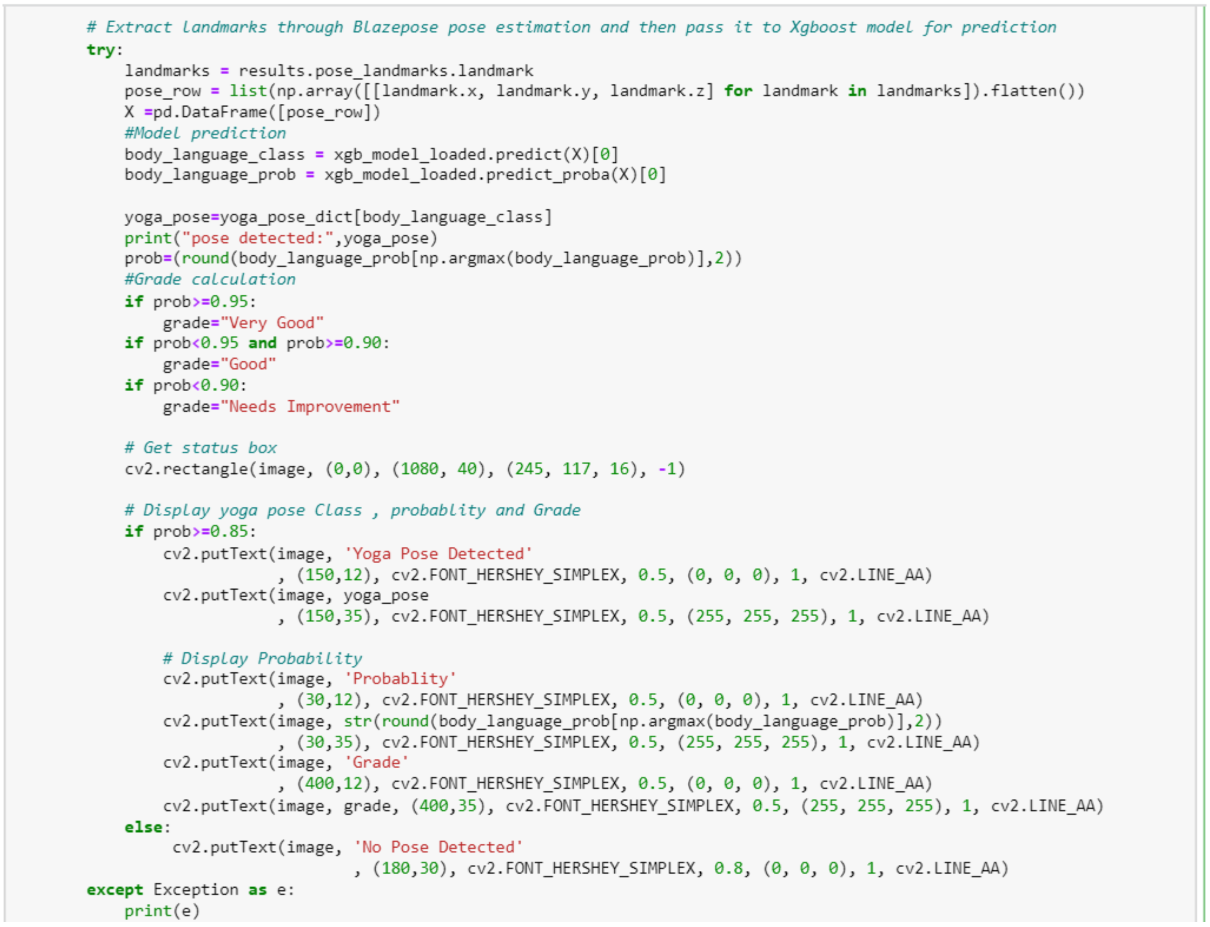
Figure 22: Yoga pose Image Detection - Part 2

Experiment 5 - Yoga Pose Detection from Real-Time Video



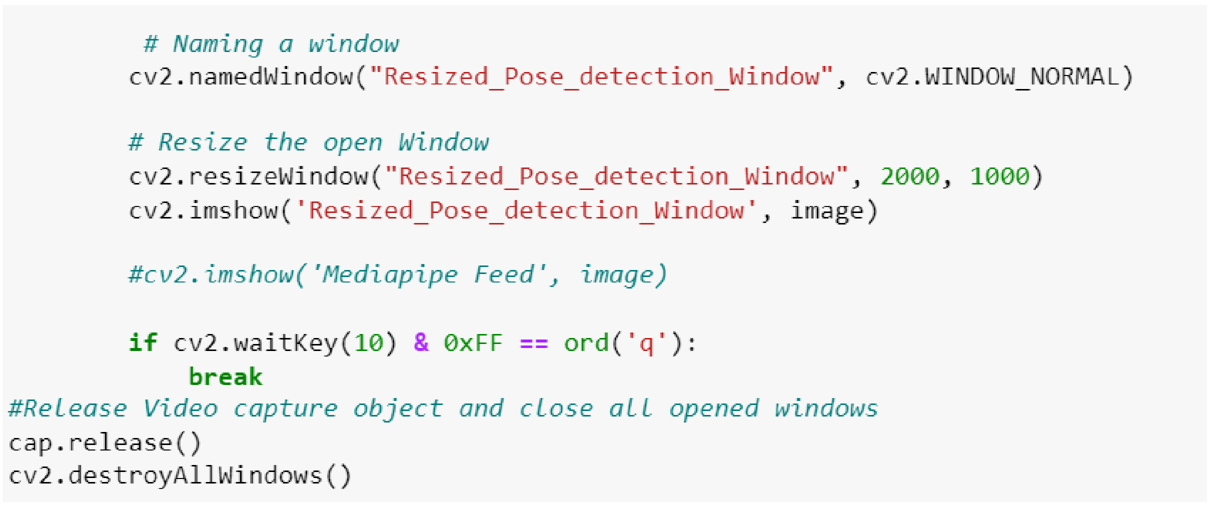
```
#Prediction from Real-time video
# Prerequistics :Make sure the webcam is working, Stand 2-3 metres away from camera,
#Make sure the entire body from head to toe is visible in the webcam
#-------------------------------------------------------------------------------------#
#Initialize the videocapture object
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        #cap.read to read each frames of the video
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection through Blazepose
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)


        # Render detections
        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)
                                 )
```

Figure 23: Real-Time Yoga pose Detection - Part 1



```
        # Extract landmarks through Blazepose pose estimation and then pass it to Xgboost model for prediction
        try:
            landmarks = results.pose_landmarks.landmark
            pose_row = list(np.array([[landmark.x, landmark.y, landmark.z] for landmark in landmarks]).flatten())
            X =pd.DataFrame([pose_row])
            #Model prediction
            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))
            #Grade calculation
            if prob>=0.95:
                grade="Very Good"
            if prob<0.95 and prob>=0.90:
                grade="Good"
            if prob<0.90:
                grade="Needs Improvement"

            # Get status box
            cv2.rectangle(image, (0,0), (1080, 40), (245, 117, 16), -1)

            # Display yoga pose Class , probablity and Grade
            if prob>=0.85:
                cv2.putText(image, 'Yoga Pose Detected'
                            , (150,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
                cv2.putText(image, yoga_pose
                            , (150,35), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv2.LINE_AA)

                # Display Probability
                cv2.putText(image, 'Probablity'
                            , (30,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 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, 'Grade'
                            , (400,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 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, (0, 0, 0), 1, cv2.LINE_AA)
        except Exception as e:
            print(e)
```

Figure 24: Real-Time Yoga pose Video Detection - Part 2



```
         # Naming a window
        cv2.namedWindow("Resized_Pose_detection_Window", cv2.WINDOW_NORMAL)

        # Resize the open Window
        cv2.resizeWindow("Resized_Pose_detection_Window", 2000, 1000)
        cv2.imshow('Resized_Pose_detection_Window', image)

        #cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
#Release Video capture object and close all opened windows
cap.release()
cv2.destroyAllWindows()
```

Figure 25: Real-Time Yoga pose Video Detection - Part 3