Configuration Manual

MSc Research Project
MSc Data Analytics

Monika Maheswari Baskar
Student ID: X22200169

School of Computing
National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Monika Maheswari Baskar		
Student ID:	X22200169		
Programme:	MSc Data Analytics	**Year:**	2023-2024
Module:	MSc Research Project		
Lecturer:	Vikas Tomer		
Submission Due Date:	16/09/2024		
Project Title:	Yoga Pose Prediction Using InceptionV3 an Transfer Learning Approach		
Word Count:	**1110**	**Page Count: 11**	

Signature: Monika Maheswari Baskar

Date: 16/09/2024

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):	

Configuration Manual

Monika Maheswari Baskar
X22200169

1 Introduction:

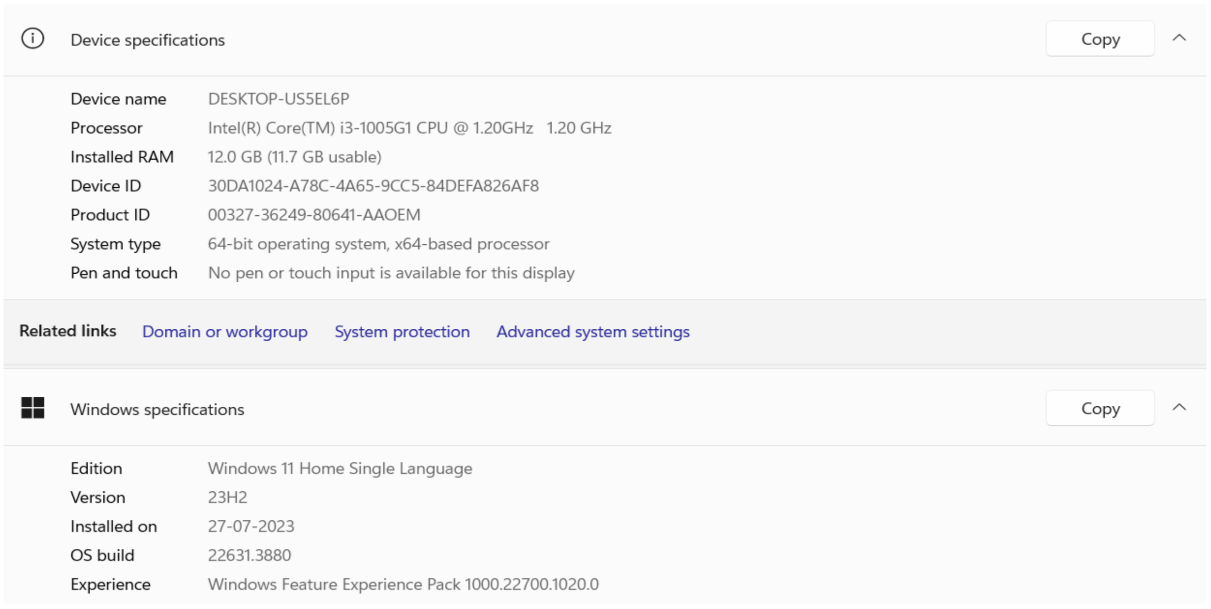
This study is titled "Yoga Pose Prediction using InceptionV3 and transfer learning approach" this research is all about predicting 38 Yoga poses. This configuration manual contains thorough information about the hardware and software to build the environment. Specific details of the data sources, system specifications, codes and libraries used for implementing and evaluating this research.

2 System Requirements:

This section explains about both the hardware and software required to implement and run the code was thoroughly described in this section. The setup for jupyter notebook through the anaconda navigator, required python libraries and packages and other components which are required are also mentioned. This section is used to explain that the code runs without any glitch and to make sure that it can be understood by everyone easily.

3 Hardware Requirements:

Figure 1 explains the hardware used for this project to understand the improved version of the hardware components have been used to run the code. Window 11 with 64 bit processor, i3 gen and 12 GB.


Device specifications

Copy

Device name	DESKTOP-US5EL6P
Processor	Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.20 GHz
Installed RAM	12.0 GB (11.7 GB usable)
Device ID	30DA1024-A78C-4A65-9CC5-84DEFA826AF8
Product ID	00327-36249-80641-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Related links Domain or workgroup System protection Advanced system settings

Windows specifications

Copy

Edition	Windows 11 Home Single Language
Version	23H2
Installed on	27-07-2023
OS build	22631.3880
Experience	Windows Feature Experience Pack 1000.22700.1020.0

Figure 1 Hardware Requirement of the system

4 Software Requirements:

Python was the only programming language used in this study for the implementation. This python code was written and executed in a jupyter notebook that was used as an easy to use platform making an excellent choice for deep learning and machine learning tasks.

- Jupyter Notebook
- Anaconda Navigator

Figure 2 shows all the necessary imported libraries needed for the research project for data pre-processing, transformation, evaluation and implementation of the InceptionV3 algorithm to implement the yoga pose prediction.

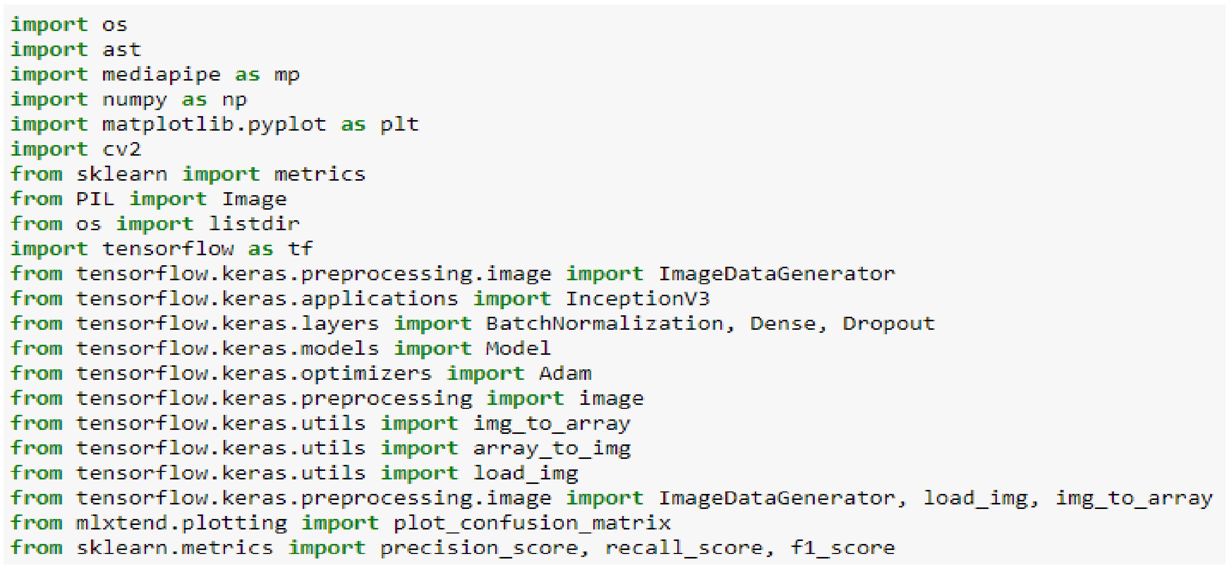

```
import os
import ast
import mediapipe as mp
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn import metrics
from PIL import Image
from os import listdir
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import BatchNormalization, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import img_to_array
from tensorflow.keras.utils import array_to_img
from tensorflow.keras.utils import load_img
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
```

Figure 2 Necessary Libraries

5 Data Collection:

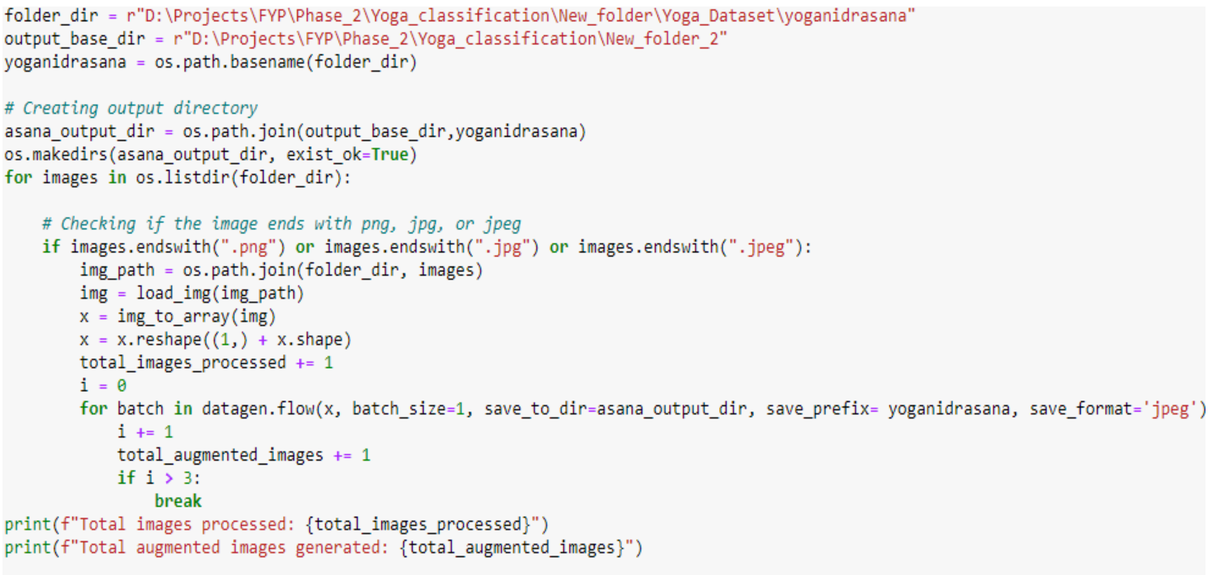
The dataset was taken from kaggle which is a publicly available dataset consisting of yoga asanas with a total number of 6720 images belonging to 38 classes which are then downloaded in the local disk and then extracted from zip to normal. Yoga poses were organized into folders with names corresponding to the respective yoga pose.

6 Implementation:

This section detail explains about the project's data, modelling, training and finally getting the results and visualizations along with the step by step guide for reproducing the study using the provided code.

7 Data Augmenting:

Open the yoga project notebook and import all the necessary packages, augmenting the images from the dataset in Figure (3) and keeping the output path as a new folder to import all the augmented images into that folder. Also while doing these changes checking if the images ends with png, jpg or jpeg after these all steps it loads and pre-processes the images and executes the result as total image processed and total augmented images.

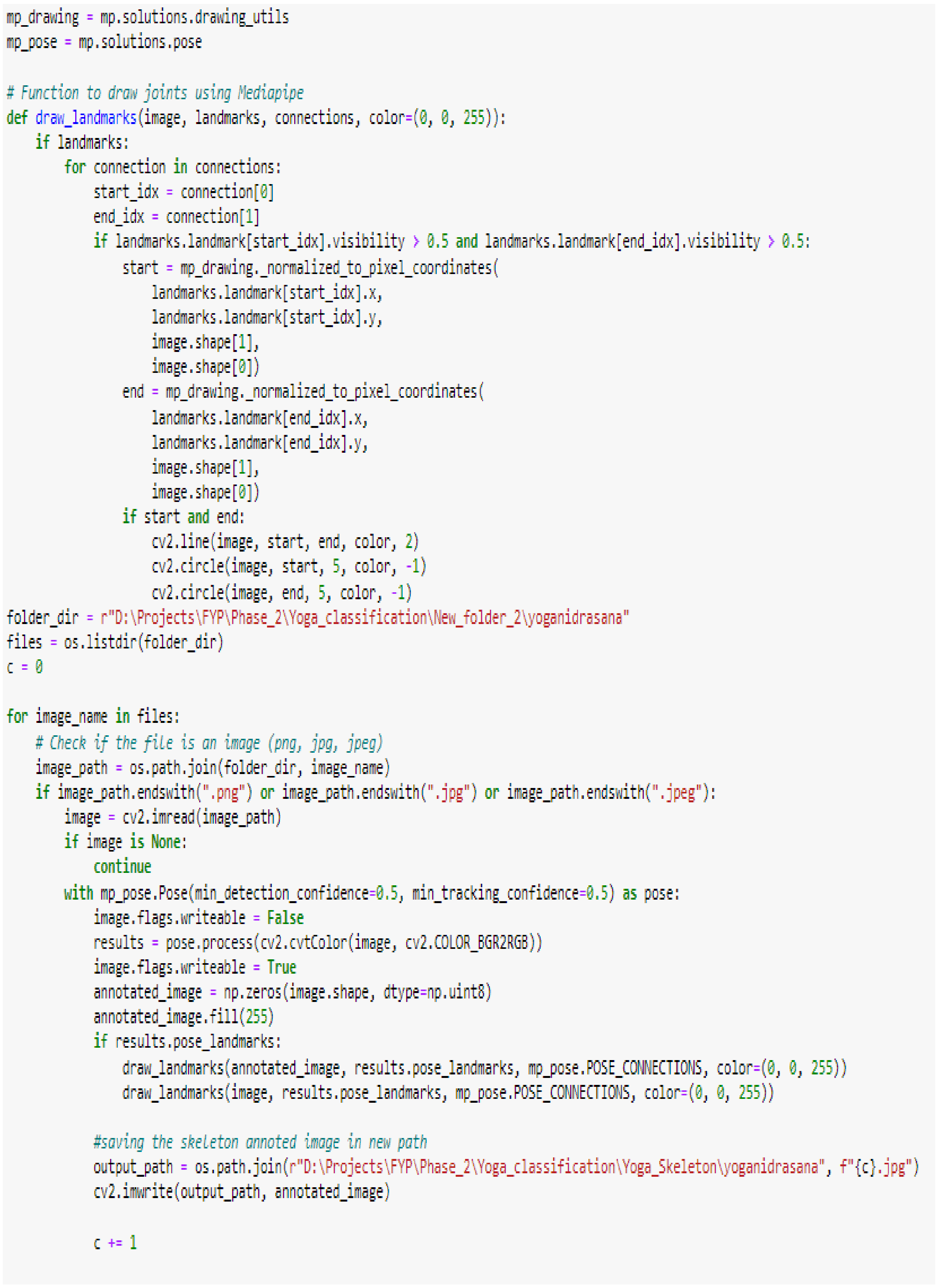


```
folder_dir = r"D:\Projects\FYP\Phase_2\Yoga_classification\New_folder\Yoga_Dataset\yoganidrasana"
output_base_dir = r"D:\Projects\FYP\Phase_2\Yoga_classification\New_folder_2"
yoganidrasana = os.path.basename(folder_dir)

# Creating output directory
asana_output_dir = os.path.join(output_base_dir,yoganidrasana)
os.makedirs(asana_output_dir, exist_ok=True)
for images in os.listdir(folder_dir):

    # Checking if the image ends with png, jpg, or jpeg
    if images.endswith(".png") or images.endswith(".jpg") or images.endswith(".jpeg"):
        img_path = os.path.join(folder_dir, images)
        img = load_img(img_path)
        x = img_to_array(img)
        x = x.reshape((1,) + x.shape)
        total_images_processed += 1
        i = 0
        for batch in datagen.flow(x, batch_size=1, save_to_dir=asana_output_dir, save_prefix= yoganidrasana, save_format='jpeg')
            i += 1
            total_augmented_images += 1
            if i > 3:
                break
print(f"Total images processed: {total_images_processed}")
print(f"Total augmented images generated: {total_augmented_images}")
```

Figure 3 augmenting the data

8 Data Preparation:

For detecting the pose and iterating through each of the image files which performs pose detection and creates the annotated images with the landmarks. The annotated images are generated and saved in an output directory Figure (4) with the filenames and this code explains that the OpenCV is used for image handling and Mediapipe for pose detection. The annotated images are the skeleton images where the original raw image changed to the skeleton format to get the precise accuracy.

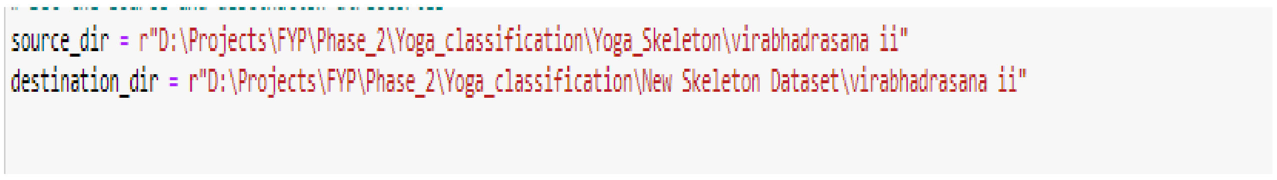


```python
mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose

# Function to draw joints using Mediapipe
def draw_landmarks(image, landmarks, connections, color=(0, 0, 255)):
    if landmarks:
        for connection in connections:
            start_idx = connection[0]
            end_idx = connection[1]
            if landmarks.landmark[start_idx].visibility > 0.5 and landmarks.landmark[end_idx].visibility > 0.5:
                start = mp_drawing._normalized_to_pixel_coordinates(
                    landmarks.landmark[start_idx].x,
                    landmarks.landmark[start_idx].y,
                    image.shape[1],
                    image.shape[0])
                end = mp_drawing._normalized_to_pixel_coordinates(
                    landmarks.landmark[end_idx].x,
                    landmarks.landmark[end_idx].y,
                    image.shape[1],
                    image.shape[0])
                if start and end:
                    cv2.line(image, start, end, color, 2)
                    cv2.circle(image, start, 5, color, -1)
                    cv2.circle(image, end, 5, color, -1)
folder_dir = r"D:\Projects\FYP\Phase_2\Yoga_classification\New_folder_2\yoganidrasana"
files = os.listdir(folder_dir)
c = 0

for image_name in files:
    # Check if the file is an image (png, jpg, jpeg)
    image_path = os.path.join(folder_dir, image_name)
    if image_path.endswith(".png") or image_path.endswith(".jpg") or image_path.endswith(".jpeg"):
        image = cv2.imread(image_path)
        if image is None:
            continue
        with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
            image.flags.writeable = False
            results = pose.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
            image.flags.writeable = True
            annotated_image = np.zeros(image.shape, dtype=np.uint8)
            annotated_image.fill(255)
            if results.pose_landmarks:
                draw_landmarks(annotated_image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS, color=(0, 0, 255))
                draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS, color=(0, 0, 255))

            #saving the skeleton annoted image in new path
            output_path = os.path.join(r"D:\Projects\FYP\Phase_2\Yoga_classification\Yoga_Skeleton\yoganidrasana", f"{c}.jpg")
            cv2.imwrite(output_path, annotated_image)

            c += 1
```

Figure 4 Pose detection and Skeleton annotation

9 Reading the data:

After data preparation the dataset has been divided into train and validation. Dataset is going to perform the prediction of the yoga asana and these 38 asanas are going to be used in model prediction as shown in Figure (5).

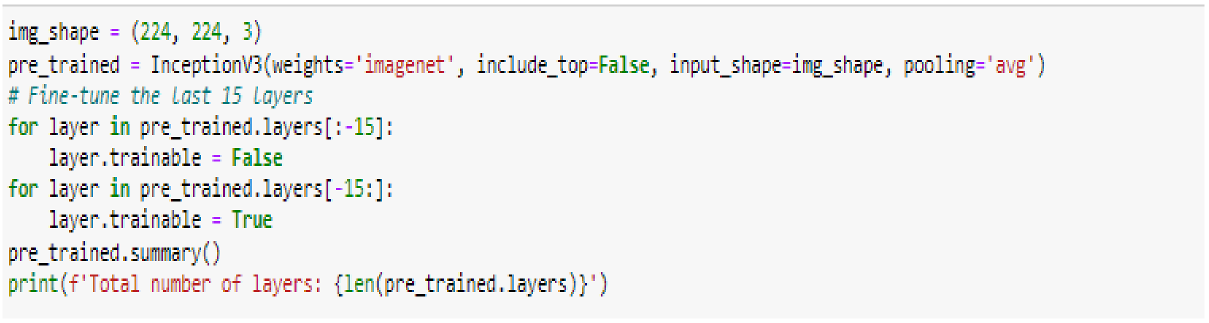


```
source_dir = r"D:\Projects\FYP\Phase_2\Yoga_classification\Yoga_Skeleton\virabhadrasana ii"
destination_dir = r"D:\Projects\FYP\Phase_2\Yoga_classification\New Skeleton Dataset\virabhadrasana ii"
```

Figure 5 Reading the data

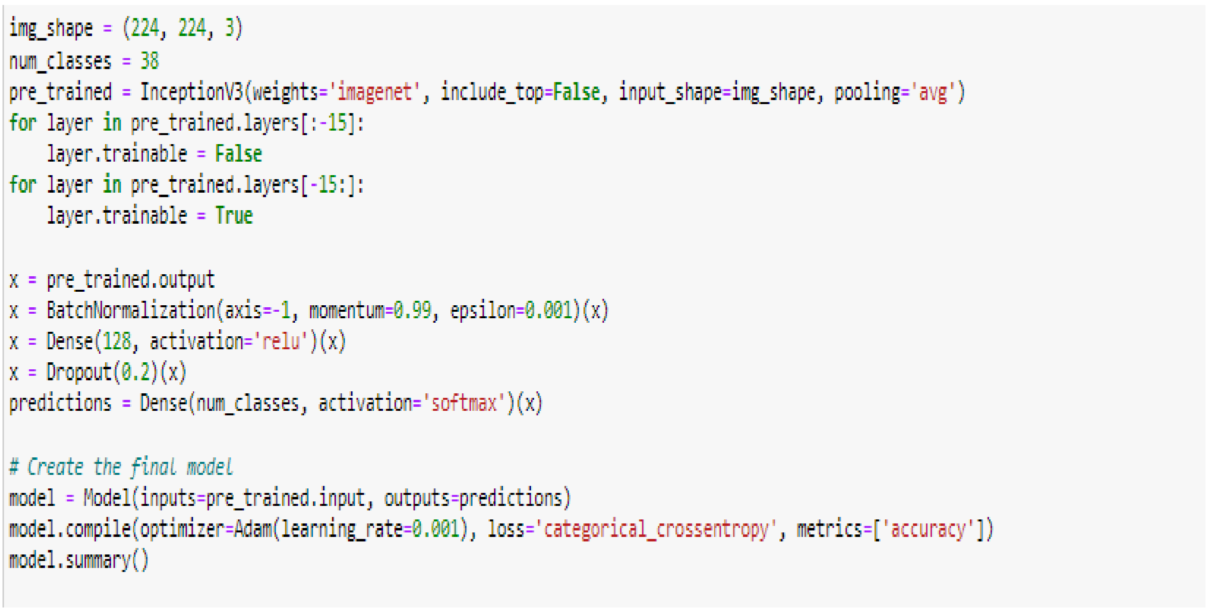
10 Model Training InceptionV3:

In the same notebook installing the InceptionV3 model as shown in figure (6) and by adding some custom layers from batch normalization, ReLU and Softmax activation as shown in figure(7).



```
img_shape = (224, 224, 3)
pre_trained = InceptionV3(weights='imagenet', include_top=False, input_shape=img_shape, pooling='avg')
# Fine-tune the last 15 layers
for layer in pre_trained.layers[:-15]:
    layer.trainable = False
for layer in pre_trained.layers[-15:]:
    layer.trainable = True
pre_trained.summary()
print(f'Total number of layers: {len(pre_trained.layers)}')
```

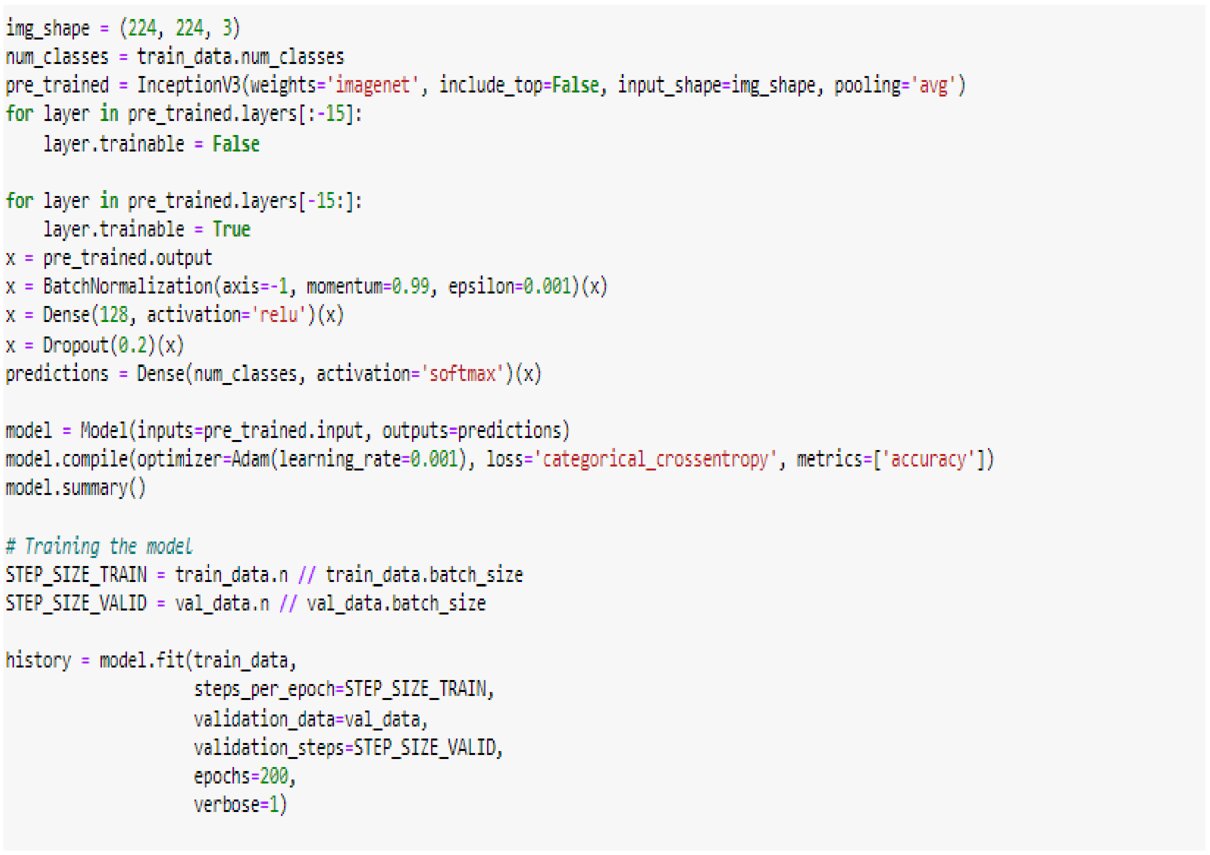
Figure 6 installing the InceptionV3 model



```
img_shape = (224, 224, 3)
num_classes = 38
pre_trained = InceptionV3(weights='imagenet', include_top=False, input_shape=img_shape, pooling='avg')
for layer in pre_trained.layers[:-15]:
    layer.trainable = False
for layer in pre_trained.layers[-15:]:
    layer.trainable = True

x = pre_trained.output
x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(num_classes, activation='softmax')(x)

# Create the final model
model = Model(inputs=pre_trained.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

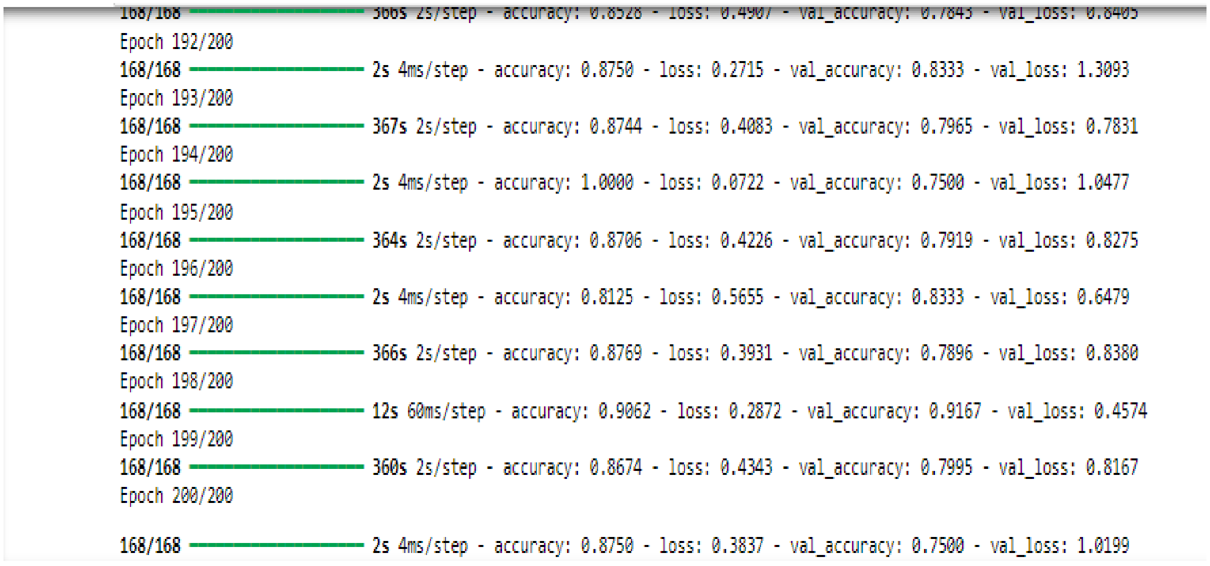
Figure 7 Adding some custom layers

After the model is trained for the architecture and established the project was trained over 200 epochs as shown in figure (8 & 9).



```python
img_shape = (224, 224, 3)
num_classes = train_data.num_classes
pre_trained = InceptionV3(weights='imagenet', include_top=False, input_shape=img_shape, pooling='avg')
for layer in pre_trained.layers[:-15]:
    layer.trainable = False

for layer in pre_trained.layers[-15:]:
    layer.trainable = True
x = pre_trained.output
x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=pre_trained.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Training the model
STEP_SIZE_TRAIN = train_data.n // train_data.batch_size
STEP_SIZE_VALID = val_data.n // val_data.batch_size

history = model.fit(train_data,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=val_data,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=200,
                    verbose=1)
```

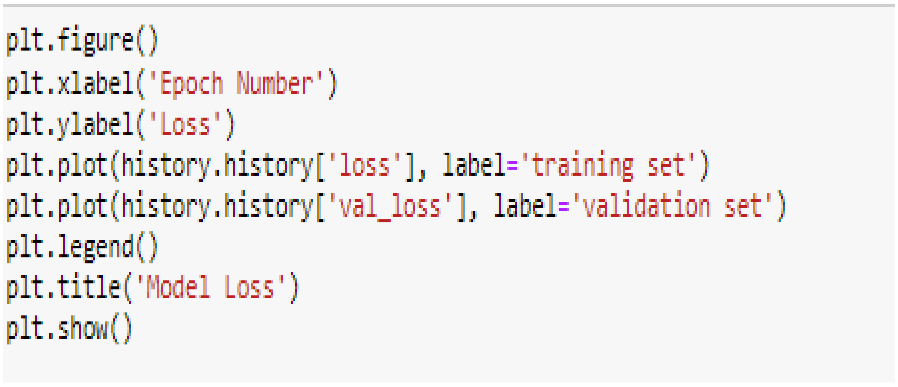
Figure 8 Adding the epoch



```
168/168 ━━━━━━━━━━━━━━━━━━━━ 366s 2s/step - accuracy: 0.8528 - loss: 0.4907 - val_accuracy: 0.7843 - val_loss: 0.8405
Epoch 192/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8750 - loss: 0.2715 - val_accuracy: 0.8333 - val_loss: 1.3093
Epoch 193/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 367s 2s/step - accuracy: 0.8744 - loss: 0.4083 - val_accuracy: 0.7965 - val_loss: 0.7831
Epoch 194/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 1.0000 - loss: 0.0722 - val_accuracy: 0.7500 - val_loss: 1.0477
Epoch 195/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 364s 2s/step - accuracy: 0.8706 - loss: 0.4226 - val_accuracy: 0.7919 - val_loss: 0.8275
Epoch 196/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8125 - loss: 0.5655 - val_accuracy: 0.8333 - val_loss: 0.6479
Epoch 197/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 366s 2s/step - accuracy: 0.8769 - loss: 0.3931 - val_accuracy: 0.7896 - val_loss: 0.8380
Epoch 198/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 12s 60ms/step - accuracy: 0.9062 - loss: 0.2872 - val_accuracy: 0.9167 - val_loss: 0.4574
Epoch 199/200
168/168 ━━━━━━━━━━━━━━━━━━━━ 360s 2s/step - accuracy: 0.8674 - loss: 0.4343 - val_accuracy: 0.7995 - val_loss: 0.8167
Epoch 200/200

168/168 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8750 - loss: 0.3837 - val_accuracy: 0.7500 - val_loss: 1.0199
```

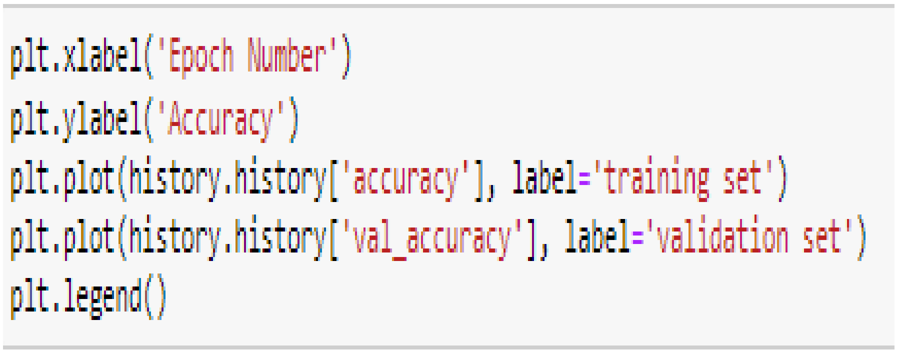
Figure 9 Result of 200 epoch

Once the model is trained in InceptionV3 the next step is to print the accuracy and loss of InceptionV3 model in Figure (10 & 11). Once the code is executed it will print the graph of both loss and accuracy in Figure (12 & 13).



```
plt.figure()
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.plot(history.history['loss'], label='training set')
plt.plot(history.history['val_loss'], label='validation set')
plt.legend()
plt.title('Model Loss')
plt.show()
```

Figure 10 Code for model loss



```
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.plot(history.history['accuracy'], label='training set')
plt.plot(history.history['val_accuracy'], label='validation set')
plt.legend()
```

Figure 11 Code for model accuracy

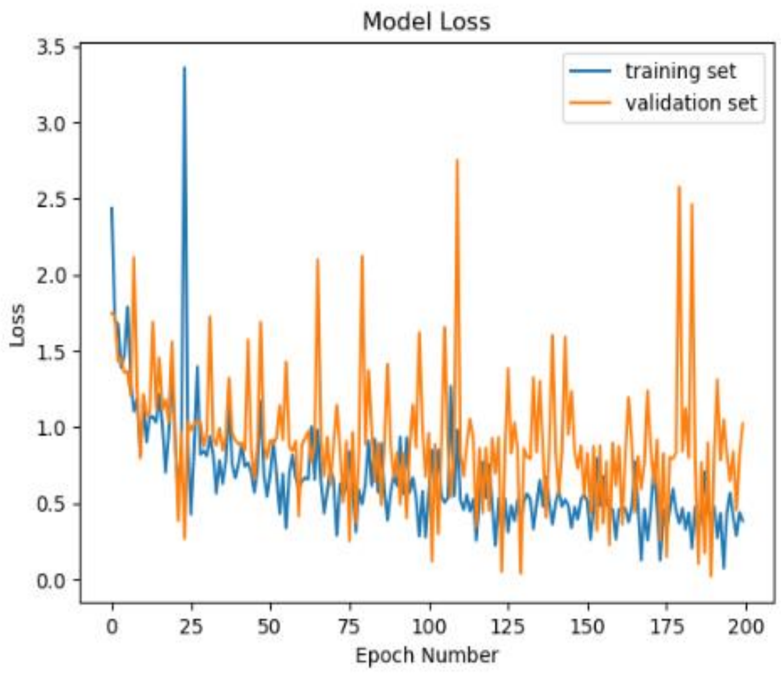


Figure 12 Graph of model loss

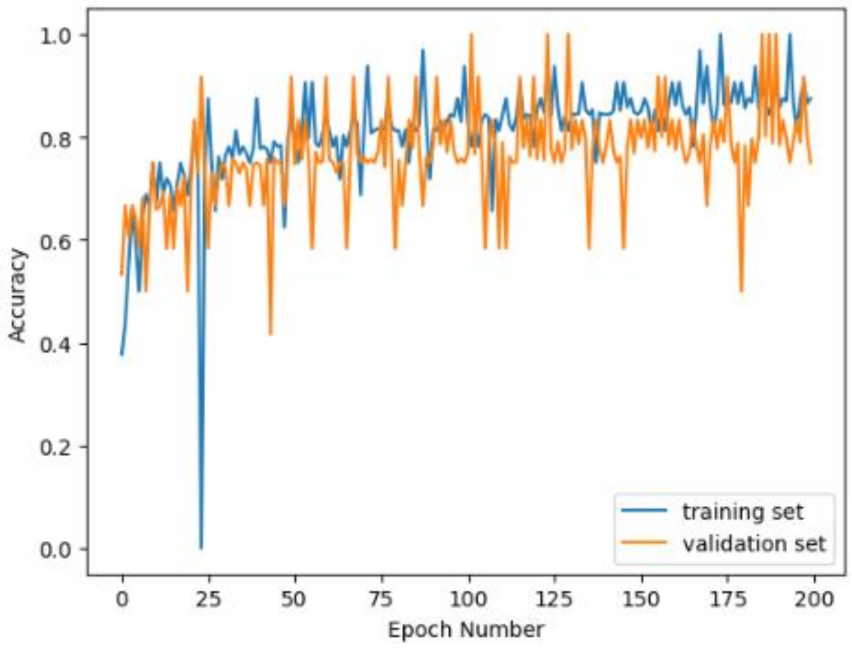
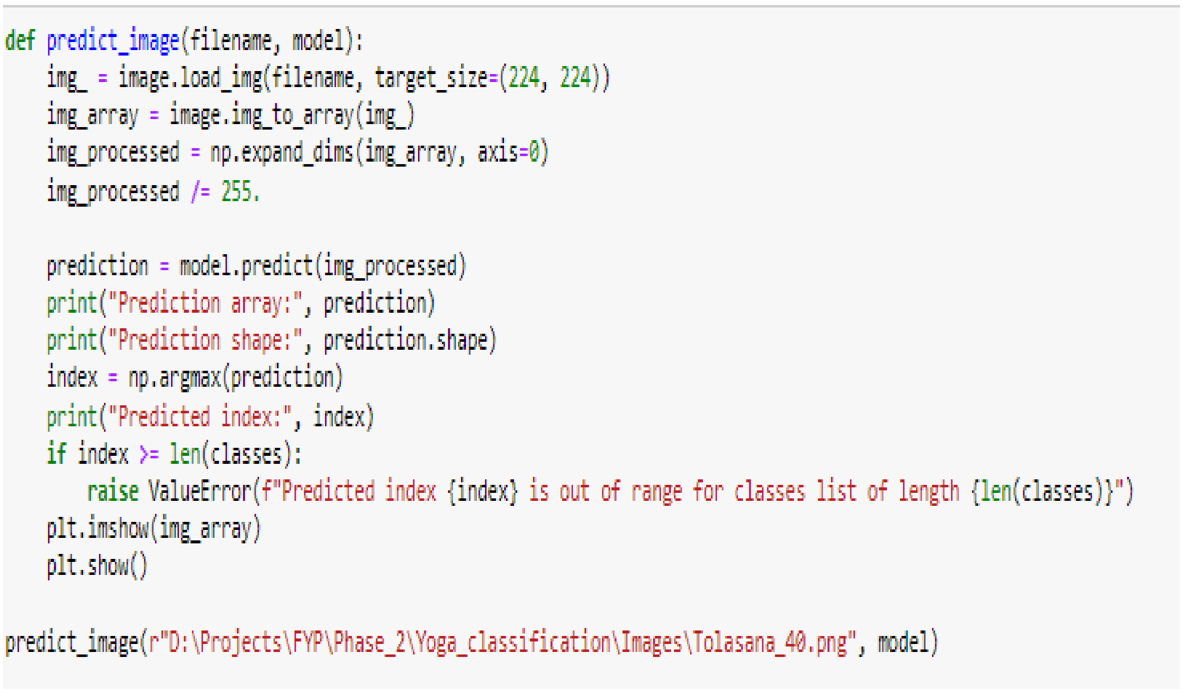


Figure 13 Graph of model accuracy

11 Model Evaluating:

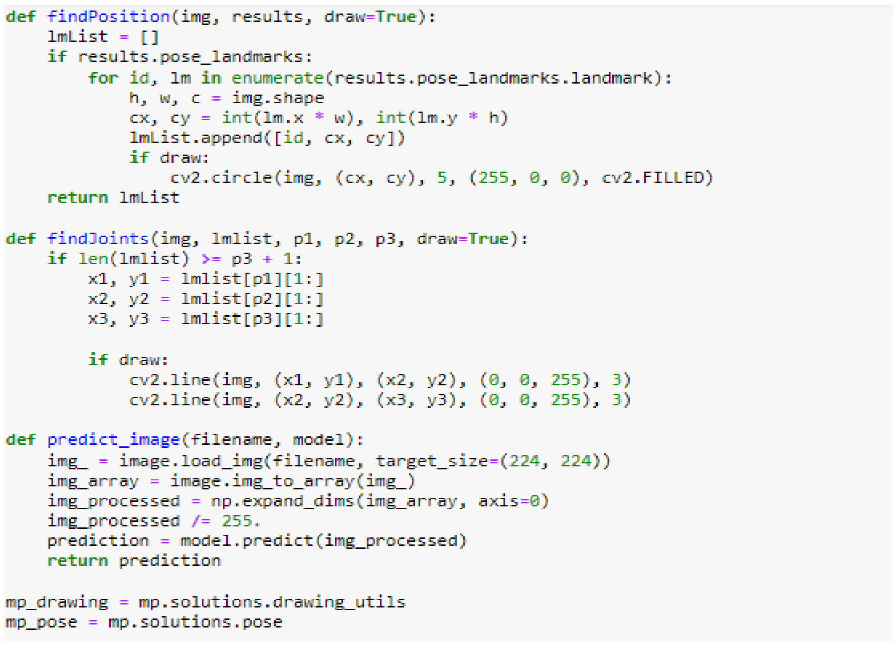
After training the model the next step is to check on the evaluation part by giving the predicted image directory path to check whether the trained model is predicted perfectly or not as shown in figure (14).



```python
def predict_image(filename, model):
    img_ = image.load_img(filename, target_size=(224, 224))
    img_array = image.img_to_array(img_)
    img_processed = np.expand_dims(img_array, axis=0)
    img_processed /= 255.

    prediction = model.predict(img_processed)
    print("Prediction array:", prediction)
    print("Prediction shape:", prediction.shape)
    index = np.argmax(prediction)
    print("Predicted index:", index)
    if index >= len(classes):
        raise ValueError(f"Predicted index {index} is out of range for classes list of length {len(classes)}")
    plt.imshow(img_array)
    plt.show()

predict_image(r"D:\Projects\FYP\Phase_2\Yoga_classification\Images\Tolasana_40.png", model)
```

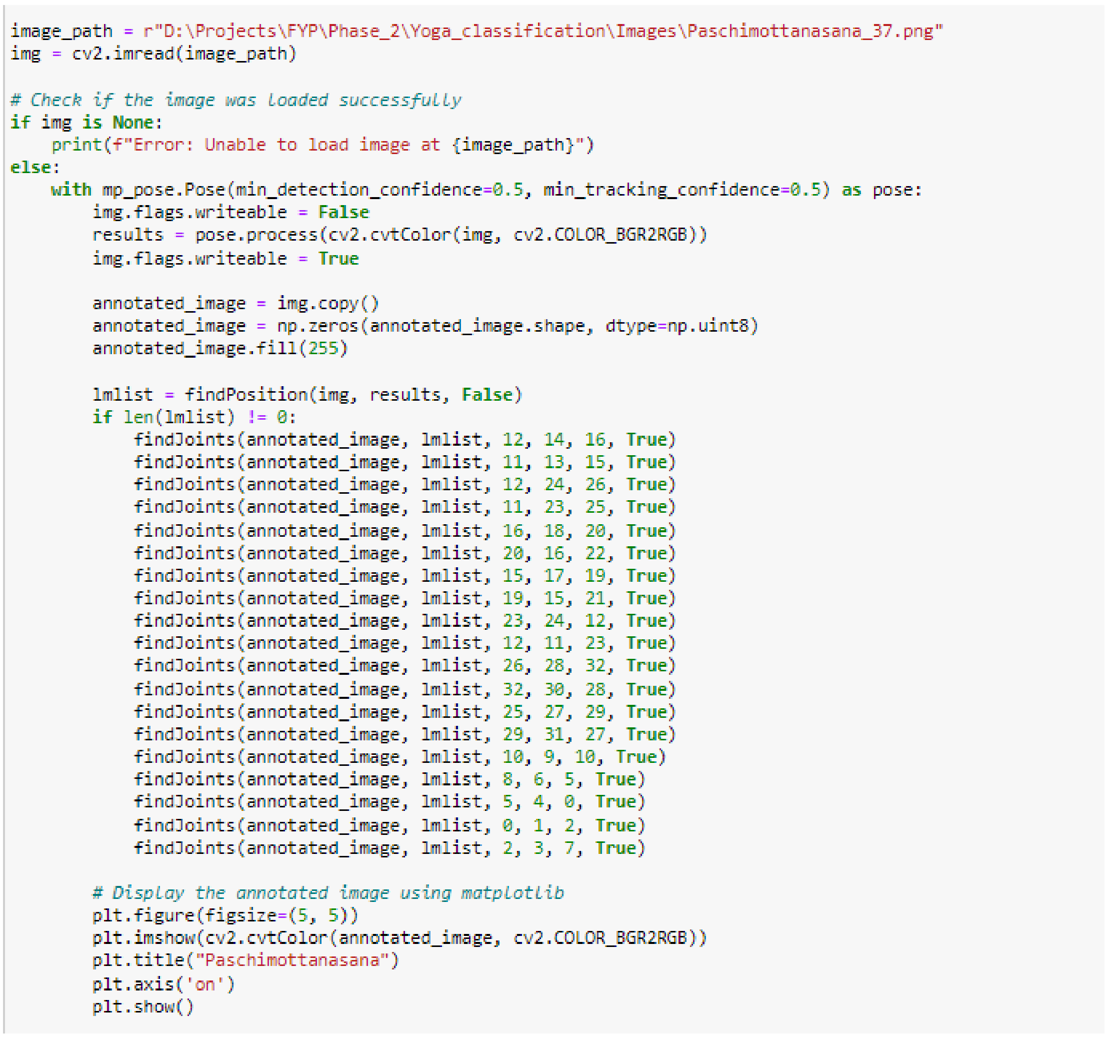
Figure 14 Checking the Imge is predicting

The final step is if we give the image path with raw images it should predict the image as the same but as the skeleton image with their respective asana name. In figure (14 & 15) it predicts and gives the Paschimottanasana asana perfectly.



```python
def findPosition(img, results, draw=True):
    lmList = []
    if results.pose_landmarks:
        for id, lm in enumerate(results.pose_landmarks.landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            lmList.append([id, cx, cy])
            if draw:
                cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)
    return lmList

def findJoints(img, lmlist, p1, p2, p3, draw=True):
    if len(lmlist) >= p3 + 1:
        x1, y1 = lmlist[p1][1:]
        x2, y2 = lmlist[p2][1:]
        x3, y3 = lmlist[p3][1:]

        if draw:
            cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 3)
            cv2.line(img, (x2, y2), (x3, y3), (0, 0, 255), 3)

def predict_image(filename, model):
    img_ = image.load_img(filename, target_size=(224, 224))
    img_array = image.img_to_array(img_)
    img_processed = np.expand_dims(img_array, axis=0)
    img_processed /= 255.
    prediction = model.predict(img_processed)
    return prediction

mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose
```

Figure 15



```python
image_path = r"D:\Projects\FYP\Phase_2\Yoga_classification\Images\Paschimottanasana_37.png"
img = cv2.imread(image_path)

# Check if the image was loaded successfully
if img is None:
    print(f"Error: Unable to load image at {image_path}")
else:
    with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
        img.flags.writeable = False
        results = pose.process(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        img.flags.writeable = True

        annotated_image = img.copy()
        annotated_image = np.zeros(annotated_image.shape, dtype=np.uint8)
        annotated_image.fill(255)

        lmlist = findPosition(img, results, False)
        if len(lmlist) != 0:
            findJoints(annotated_image, lmlist, 12, 14, 16, True)
            findJoints(annotated_image, lmlist, 11, 13, 15, True)
            findJoints(annotated_image, lmlist, 12, 24, 26, True)
            findJoints(annotated_image, lmlist, 11, 23, 25, True)
            findJoints(annotated_image, lmlist, 16, 18, 20, True)
            findJoints(annotated_image, lmlist, 20, 16, 22, True)
            findJoints(annotated_image, lmlist, 15, 17, 19, True)
            findJoints(annotated_image, lmlist, 19, 15, 21, True)
            findJoints(annotated_image, lmlist, 23, 24, 12, True)
            findJoints(annotated_image, lmlist, 12, 11, 23, True)
            findJoints(annotated_image, lmlist, 26, 28, 32, True)
            findJoints(annotated_image, lmlist, 32, 30, 28, True)
            findJoints(annotated_image, lmlist, 25, 27, 29, True)
            findJoints(annotated_image, lmlist, 29, 31, 27, True)
            findJoints(annotated_image, lmlist, 10, 9, 10, True)
            findJoints(annotated_image, lmlist, 8, 6, 5, True)
            findJoints(annotated_image, lmlist, 5, 4, 0, True)
            findJoints(annotated_image, lmlist, 0, 1, 2, True)
            findJoints(annotated_image, lmlist, 2, 3, 7, True)

        # Display the annotated image using matplotlib
        plt.figure(figsize=(5, 5))
        plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))
        plt.title("Paschimottanasana")
        plt.axis('on')
        plt.show()
```

Figure 16 Predicting the training data

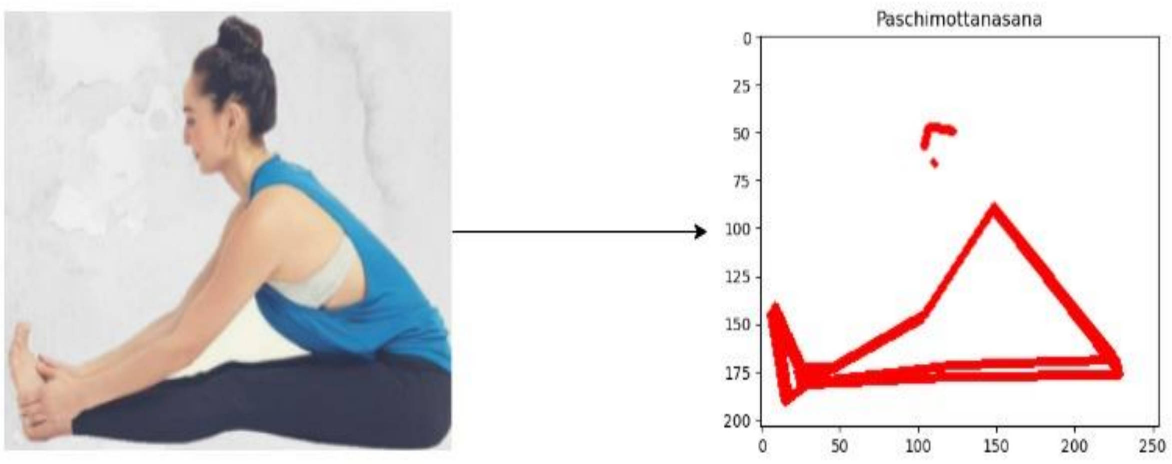


Figure 17 Final Output

12 Reference:

Dataset Link : Yoga Pose Image classification Dataset Available :
https://www.kaggle.com/datasets/shrutisaxena/yoga-pose-image-classification-dataset