Yoga Pose Multiclass Classification Using Machine Learning Models
Configuration Manual

MSc Research Project
Data Analytics

Aishwarya Ubale
Student ID: x22112081

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aishwarya Ubale		
Student ID:	X22112081		
Programme:	Data Analytics	**Year:**	2023-24
Module:	MSc Research Project		
Lecturer:	Christian Horn		
Submission Due Date:	31/01/2024		
Project Title:	Yoga Pose Multiclass Classification using Machine Learning Models		
Word Count:	822	**Page Count: 22**	

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: Aishwarya Ubale

Date: 31/01/2024

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):	

Yoga Pose Classification through Default and Optimized Machine Learning Techniques Configuration Manual

Aishwarya Ubale
Student ID: x22112081

1 Introduction

This document provides detailed instructions for how to setup the system or device for successful execution of the project – 'Yoga Pose Multiclass classification using Machine learning models'. The main motive is to give an outline of the research study. Machine configurations needed for model building and execution are also explained in this document. Required applications and packages needed as a starting point setup is also given in the configuration manual.

2 System Specifications

- **Hardware Specifications:**

  This is a written description of the system requirements needed for smooth execution of the project. Figure 2 depicts the system requirements used to run this project. Minimum requirement is to have 8GB RAM to successfully execute the code.

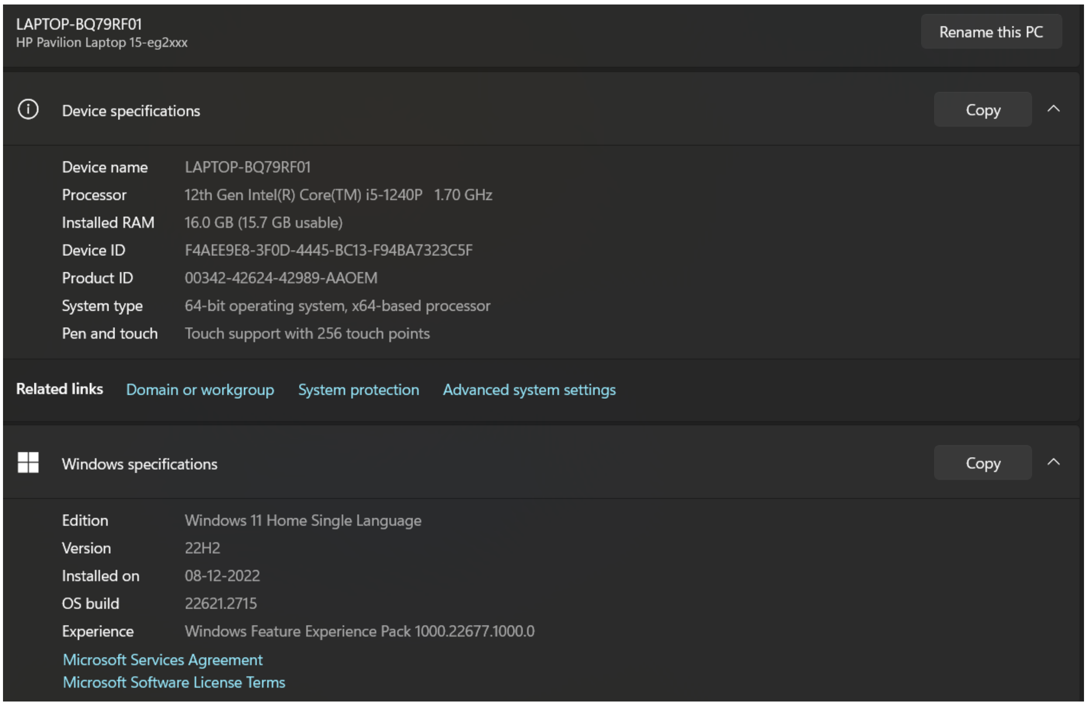


Figure 1 System Specification

1. **Software Specifications**
    - **Microsoft Excel : For initial data exploration**
    - **Jupyter Notebook : For model building and evaluation purpose.**

3 Download and install necessary packages:

Python is fundamental package that is to be installed depending on the base operating system. It is recommended to have latest version of Python[1]. For Windows 11 , version – Python 3.10.9 was downloaded and used. For executing the python files, an environment is needed. The Jupyter notebook is most commonly used platform, that has been used for this project. Anaconda [2] has been downloaded and installed. After successful installation of Jupyter notebook, to start the development using Python code, it is necessary to launch the Jupyter notebook and create a python (.ipynb) file.

4 Development of the Project:

Completion of the installing required packages and softwares, the user will have to launch the Jupyter notebook , click NEW in the left side of the notebook that will open new .ipynb Python notebook ready for the code development. Using RUN button, user can run individual cell or run all the cells by clicking on RUN ALL button. The following command will be used to install any necessary libraries needed – ‘pip install package-name’ on command prompt of the Jupyter Notebook.

4.1 Importing Libraries:

[1] https://www.python.org/downloads/

[2] https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/

The following image depicts all the required libraries needed to be installed for the successful execution of the code.

```
import mediapipe as mp
import cv2
import time
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score,classification_report
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score, f1_score, cohen_kappa_score
from sklearn.model_selection import StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from keras.preprocessing.image import ImageDataGenerator
from PIL import Image
import random
```

Figure 2 Packages utilized in the project.

4.2 Dataset Description:

The dataset is downloaded from Kaggle website [3]. The dataset has two subfolders – Train and Test data having five poses – Tree, Downdog, Warrior, Goddess, Plank. In all 1542 images are present in the dataset folder. Following image – figure 3 depicts the code to access the folder and display first image from each class. The images are displayed in figure 4. The dataset folder and the (.ipynb file) should be placed inside one folder.

[3] https://www.kaggle.com/datasets/niharika41298/yoga-poses-dataset/data

```python
import os
import matplotlib.pyplot as plt
from PIL import Image

def display_first_images_in_grid(class_directory):
    class_folders = sorted(os.listdir(class_directory))
    num_classes = len(class_folders)

    rows = 2
    cols = (num_classes + rows - 1) // rows

    fig, axes = plt.subplots(rows, cols, figsize=(12, 6))

    if isinstance(axes, np.ndarray):
        axes = axes.flatten()
    else:
        axes = [axes]

    for i, class_folder in enumerate(class_folders):
        class_path = os.path.join(class_directory, class_folder)
        image_files = os.listdir(class_path)

        if len(image_files) > 0:
            first_image = image_files[0]  # Select the first image
            image_path = os.path.join(class_path, first_image)
            img = Image.open(image_path)

            axes[i].imshow(img)
            axes[i].set_title(f"Class: {class_folder}")
            axes[i].axis('off')

    for j in range(num_classes, rows * cols):
        axes[j].axis('off')
        axes[j].set_visible(False)

    plt.tight_layout()
    plt.show()

train_directory = "archive (6)\\DATASET\\TRAIN"
display_first_images_in_grid(train_directory)
```

Figure 3 Code for accessing dataset folder.

Figure 4 Sample Images from each class

4.3 Default parameter modelling:

4.3.1 Landmark Detection – Original Dataset

```
path = "archive (6)\\DATASET\\TEST"

count = 0
data.drop(data.index, inplace=True)
counterfolder = 0;
for imgfolder in os.listdir(path):
    print(imgfolder)
    imgpath = path + "//" + imgfolder
    for img in os.listdir(imgpath):

        temp = []
        #print(img)
        img = cv2.imread(imgpath + "/" + img)

        imageWidth, imageHeight = img.shape[:2]

        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        #blackie = np.zeros(img.shape) # Blank image
        blackie = np.zeros(img.shape, dtype=np.float32)

        results = pose.process(imgRGB)

        if results.pose_landmarks:

                mpDraw.draw_landmarks(blackie, results.pose_landmarks, mpPose.POSE_CONNECTIONS) # draw landmarks on blackie

                landmarks = results.pose_landmarks.landmark

                for i,j in zip(points,landmarks):

                        temp = temp + [j.x, j.y, j.z, j.visibility]

                #Assign target value (0 for the first pose, 1 for the rest)
                if imgfolder == "plank":
                    target = 0
                elif imgfolder == "warrior2":
                    target = 1
                elif imgfolder == "tree":
                    target = 2
                elif imgfolder == "downdog":
                    target = 3
                elif imgfolder == "goddess":
                    target = 4

                temp.append(target)

                data.loc[count] = temp

                count +=1

data.to_csv("testing_dataset3_with target_allposses.csv") # save the data as a csv file
```

4.3.2 Model Building Using Extracted Landmarks Dataset:

```
In [11]:  #Read the training dataset:
          from sklearn.model_selection import train_test_split
          from sklearn.svm import SVC
          data = pd.read_csv("training_dataset_with target_all3posses.csv")
          X_train,y_train = data.iloc[:,:132],data['target']
```

```
In [13]:  #Read the dataset

          test_data = pd.read_csv("testing_dataset3_with target_allposses.csv")

In [ ]:   X_test,Y_test = test_data.iloc[:,:132],test_data['target']
```

- Support Vector Classifier:

SVC Model

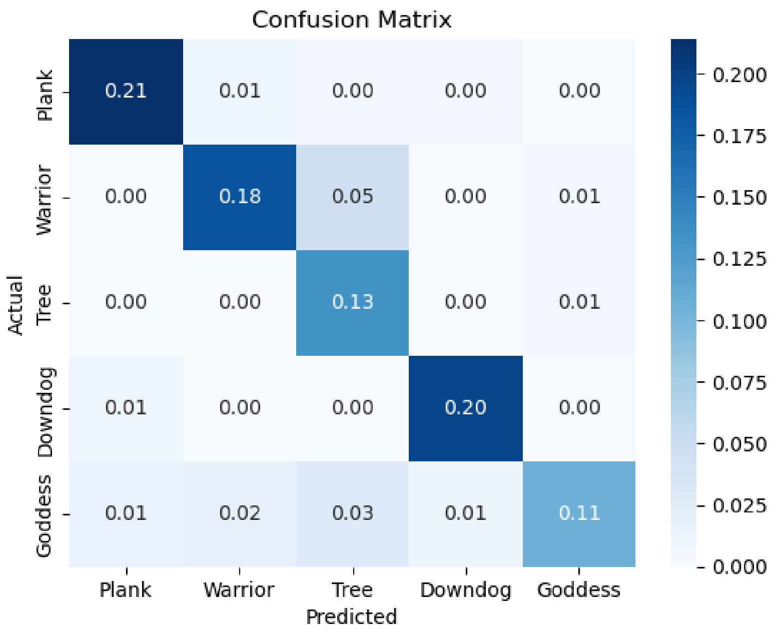
```
model = SVC(kernel='poly',class_weight='balanced', probability=True,random_state=42)
model.fit(X_train,y_train)

predictions = model.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
base_model_accuracy['SVC'] = accuracy
```

```
Accuracy: 0.8344827586206897
```

```
conf_matrix = confusion_matrix(Y_test, predictions, normalize='all')
print("Confusion Matrix as Percentages:")
print(conf_matrix)
```

```
Confusion Matrix as Percentages:
[[0.2137931  0.0091954  0.0045977  0.0045977  0.00229885]
 [0.00229885 0.18390805 0.04597701 0.         0.00689655]
 [0.00229885 0.         0.13333333 0.         0.00689655]
 [0.0091954  0.         0.         0.19770115 0.00229885]
 [0.0137931  0.01609195 0.02758621 0.01149425 0.10574713]]
```

```
# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True,fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



- Random Forest Classifier:

Model 2 - Random Forest

-
```python
from sklearn.ensemble import RandomForestClassifier

# Create and train the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train,y_train)
```

```
0]: RandomForestClassifier(random_state=42)
```

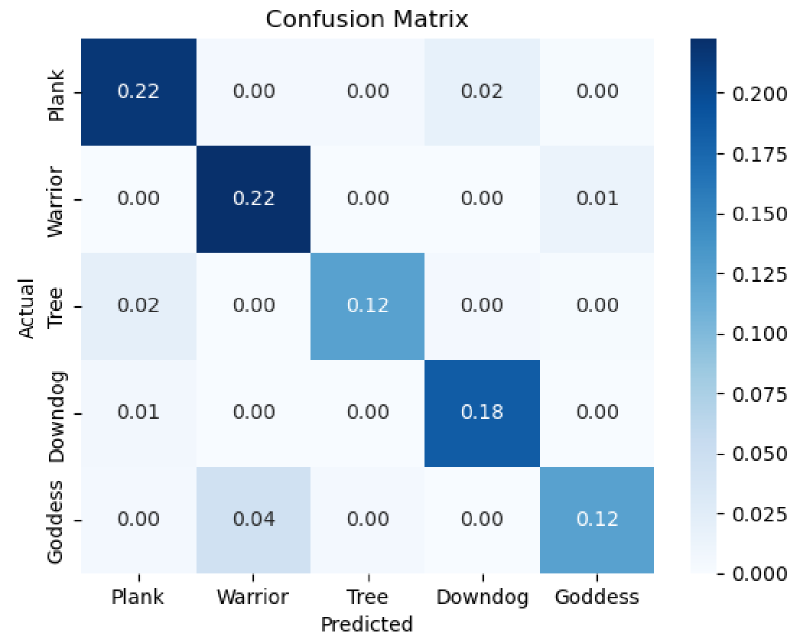
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

-
```python
predictions = model.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
base_model_accuracy['RFC'] = accuracy
```

```
Accuracy: 0.8206896551724138
```

-
```python
conf_matrix = metrics.confusion_matrix(Y_test, predictions, normalize='all')

print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.21573034 0.00449438 0.00449438 0.01797753 0.00224719]
 [0.         0.22247191 0.         0.         0.01348315]
 [0.02022472 0.         0.12359551 0.00449438 0.00224719]
 [0.00674157 0.         0.         0.18426966 0.        ]
 [0.00449438 0.04494382 0.00449438 0.         0.12359551]]
```



- Decision Tree Classifier:

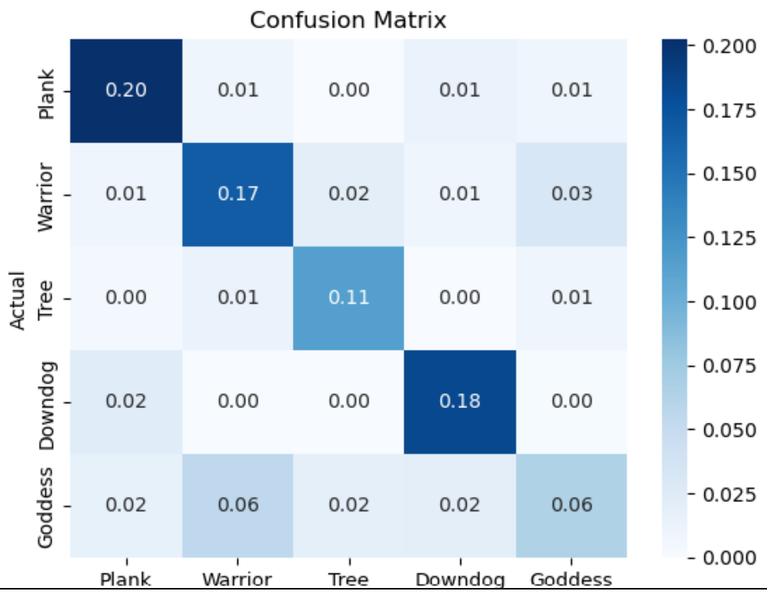
Decision Tree Classifier

```python
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train,y_train)

# Predict on the test set
predictions = clf.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
base_model_accuracy['DTC'] = accuracy
```

```
Accuracy: 0.7333333333333333
```

```python
# Calculate the raw confusion matrix
conf_matrix = confusion_matrix(Y_test, predictions, normalize='all')

print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f",cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.20229885 0.0091954  0.         0.0137931  0.0091954 ]
 [0.0091954  0.16781609 0.02068966 0.0091954  0.03218391]
 [0.0045977  0.0137931  0.11494253 0.         0.0091954 ]
 [0.02298851 0.         0.         0.18390805 0.00229885]
 [0.01609195 0.05517241 0.0183908  0.02068966 0.06436782]]
```



- K-Nearest Neighbor Classifier:

Model 3- KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier

# Create and train the KNN Classifier
np.random.seed(42)
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)
```

```
7]: KNeighborsClassifier()
```

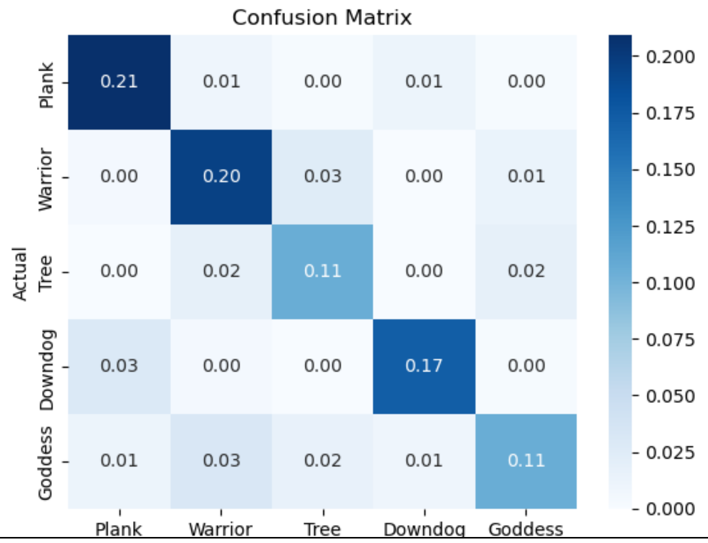
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
predictions = model.predict(X_test)
#actual_labels = test_data['target']

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
base_model_accuracy['KNN'] = accuracy
```

```
Accuracy: 0.7885057471264367
```

```
# Calculate the raw confusion matrix
conf_matrix = confusion_matrix(Y_test, predictions, normalize='all')

print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.2091954  0.0091954  0.00229885 0.01149425 0.00229885]
 [0.0045977  0.1954023  0.02528736 0.         0.0137931 ]
 [0.         0.0183908  0.10574713 0.         0.0183908 ]
 [0.02988506 0.0045977  0.         0.17241379 0.00229885]
 [0.01149425 0.03218391 0.01609195 0.0091954  0.10574713]]
```



- Gradient Boost Classifier:

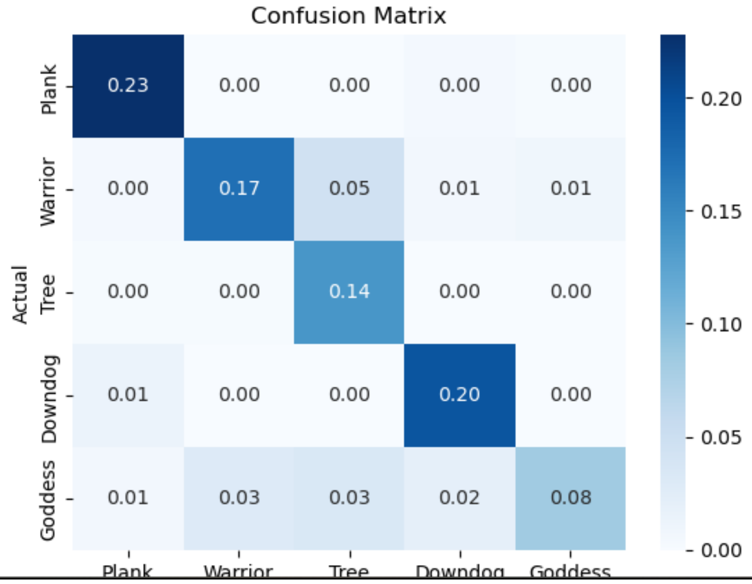
Model 5 - Gradient Boost Algorithm

```
from sklearn.ensemble import GradientBoostingClassifier

# Create and train the Gradient Boosting Classifier
model = GradientBoostingClassifier(random_state=42)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
#actual_labels = test_data['target']

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
base_model_accuracy['GBC'] = accuracy
```

```
Accuracy: 0.8160919540229885
```

```
# Calculate the raw confusion matrix
conf_matrix = confusion_matrix(Y_test, predictions, normalize='all')


print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.22758621 0.         0.         0.0045977  0.00229885]
 [0.0045977  0.17241379 0.04597701 0.00689655 0.0091954 ]
 [0.00229885 0.00229885 0.13793103 0.         0.        ]
 [0.0137931  0.         0.         0.1954023  0.        ]
 [0.00689655 0.02988506 0.03448276 0.02068966 0.08275862]]
```



Optimized parameter modelling:

4.3.3 Image Augmentation using Original Dataset

```python
import os
from keras.preprocessing.image import ImageDataGenerator

# Define paths
original_dataset_path = 'archive (6)\\DATASET\\TEST_tobeaugmented\\'  # Replace with your original dataset path
augmented_images_path = 'archive (6)\\DATASET\\TEST_ENTIRESET_AUGMENTED' # Replace with the path for augmentation

# Define augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    zoom_range=0.1,
    fill_mode='nearest'
)



# Loop through each class directory in the original dataset
for class_folder in os.listdir(original_dataset_path):
    class_folder_path = os.path.join(original_dataset_path, class_folder)
    print(class_folder)
    print(class_folder_path)
    files = os.listdir(class_folder_path + '\\' + class_folder)

    # Specify the number of images you want to generate for this class
    num_augmented_images = len(files)  # Change this based on your requirement
    print(num_augmented_images)

    # Create a directory to save augmented images for this class
    augmented_class_path = os.path.join(augmented_images_path, class_folder)
    os.makedirs(augmented_class_path, exist_ok=True)

    # Create a flow from directory generator to read images from this class
    image_generator = datagen.flow_from_directory(
        class_folder_path,
        target_size=(224, 224),  # Adjust the target size of your images
        batch_size=1,
        class_mode='categorical',
        save_to_dir=augmented_class_path,  # Directory to save augmented images for this class
        save_prefix='augmented',  # Prefix for saved images
        save_format='jpg'  # Save images in JPG format
    )

    files = os.listdir(class_folder_path + '\\' + class_folder)

    # Specify the number of images you want to generate for this class
    num_augmented_images = len(files)  # Change this based on your requirement

    # Generate augmented images and save them for this class
    for i in range(num_augmented_images):
        batch = image_generator.next()  # Generates a batch of augmented images
```

4.3.4 Landmark Detection Using Augmented Dataset

```
path = "archive (6)\\DATASET\\TEST_ENTIRESET_AUGMENTED\\"

count = 0
data.drop(data.index, inplace=True)
counterfolder = 0;
for imgfolder in os.listdir(path):
    print(imgfolder)
    imgpath = path + "//" + imgfolder
    for img in os.listdir(imgpath):

        temp = []
        #print(img)
        img = cv2.imread(imgpath + "/" + img)

        imageWidth, imageHeight = img.shape[:2]

        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        #blackie = np.zeros(img.shape) # Blank image
        blackie = np.zeros(img.shape, dtype=np.float32)

        results = pose.process(imgRGB)

        if results.pose_landmarks:

                mpDraw.draw_landmarks(blackie, results.pose_landmarks, mpPose.POSE_CONNECTIONS) # draw landmarks on blackie

                landmarks = results.pose_landmarks.landmark

                for i,j in zip(points,landmarks):

                        temp = temp + [j.x, j.y, j.z, j.visibility]

                #Assign target value (0 for the first pose, 1 for the rest)
                if imgfolder == "plank":
                    target = 0
                elif imgfolder == "warrior2":
                    target = 1
                elif imgfolder == "tree":
                    target = 2
                elif imgfolder == "downdog":
                    target = 3
                elif imgfolder == "goddess":
                    target = 4

                temp.append(target)

                data.loc[count] = temp

                count +=1

#data.to_csv("testing_dataset3_with target_allposses.csv") # save the data as a csv file
data.to_csv("testing_dataset3_with target_allposses_augmented.csv") # save the data as a csv file
```

4.3.5 Model Building Using Augmented & Landmark detected dataset.

```
#Read the training dataset:
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
data = pd.read_csv("training_dataset_with target_all3posses_augmented.csv")
X_train,y_train= data.iloc[:,:132],data['target']
```

```
#Read the test dataset


test_data = pd.read_csv("testing_dataset3_with target_allposses_augmented.csv")
#X_test = test_data.iloc[:, :132]

X_test,Y_test = test_data.iloc[:,:132],test_data['target']
```

Following are the models implemented using above obtained landmark data based on augmented images.

- Support Vector Classifier:

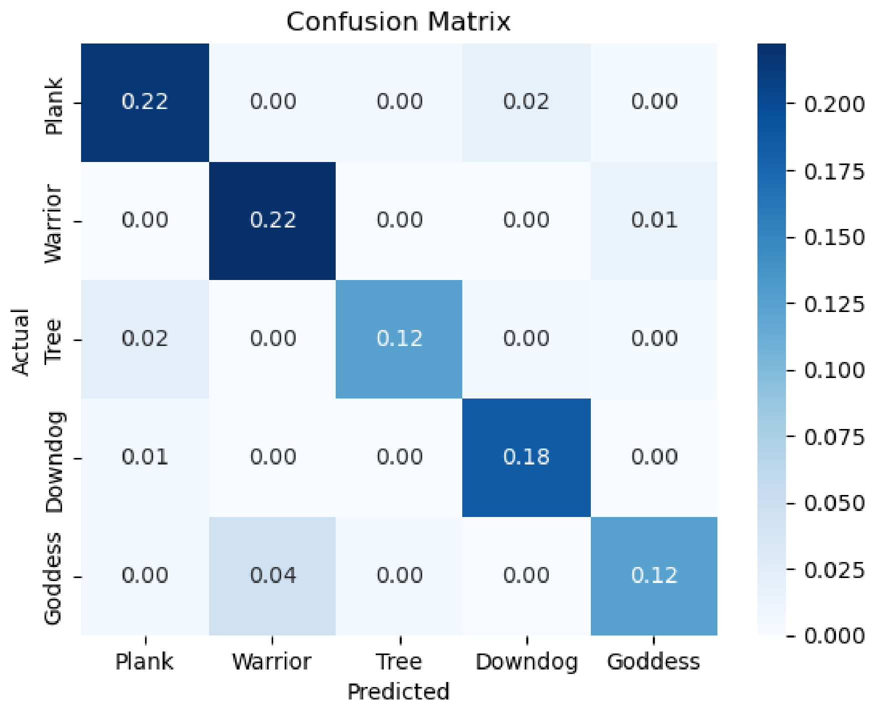
SVC Model

```python
from sklearn.svm import SVC

model = SVC(kernel='poly',probability=True,random_state=42)
model.fit(X_train,y_train)

predictions = model.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
optmized_model_accuracy['SVC'] = accuracy
```

```
Accuracy: 0.8696629213483146
```

```python
conf_matrix = metrics.confusion_matrix(Y_test, predictions, normalize='all')

print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.21573034 0.00449438 0.00449438 0.01797753 0.00224719]
 [0.         0.22247191 0.         0.         0.01348315]
 [0.02022472 0.         0.12359551 0.00449438 0.00224719]
 [0.00674157 0.         0.         0.18426966 0.        ]
 [0.00449438 0.04494382 0.00449438 0.         0.12359551]]
```



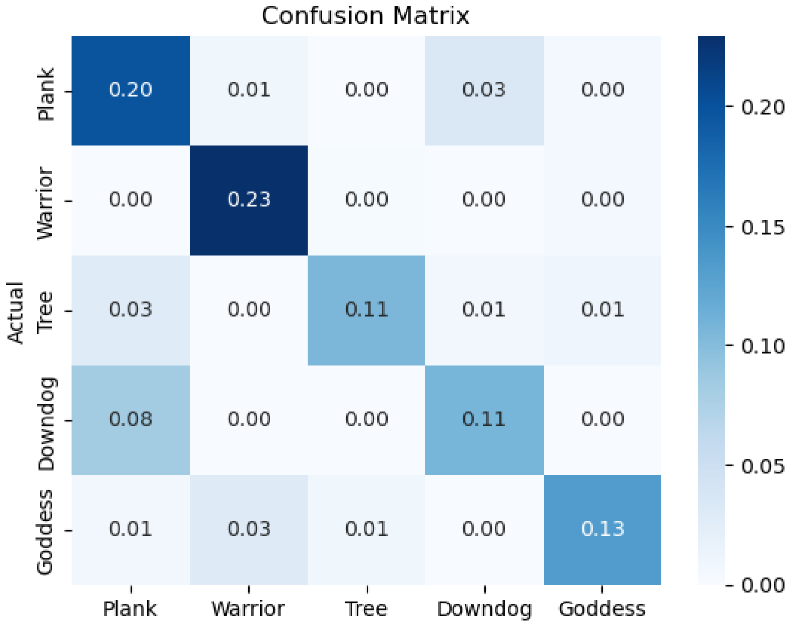
- Random Forest Classifier:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    'n_estimators': [150, 200, 250, 300],
    'max_depth': [20, 25, 30, 35],
    'min_samples_split': [2, 3, 4, 5],
    'min_samples_leaf': [1, 2, 3, 4, 5]
}


random_search = RandomizedSearchCV(model, param_distributions=param_dist, n_iter=10, cv=5, scoring='accuracy', random_state=4
random_search.fit(X_train, y_train)

best_params = random_search.best_params_
best_score = random_search.best_score_

####

# Train a new model using the best hyperparameters
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train, y_train)
predictions = best_model.predict(X_test)

# Evaluate the best model on test data
test_accuracy = best_model.score(X_test, Y_test)

print("Best hyperparameters:", best_params)
print("Best cross-validation score:", best_score)
print("Test set accuracy with best hyperparameters:", test_accuracy)
```

```
Best hyperparameters: {'n_estimators': 250, 'min_samples_split': 4, 'min_samples_leaf': 2, 'max_depth': 30}
Best cross-validation score: 0.8469165059244972
Test set accuracy with best hyperparameters: 0.7730337078651686
```

```
# Calculate the raw confusion matrix
conf_matrix = metrics.confusion_matrix(Y_test, predictions, normalize='all')


print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.19775281 0.00898876 0.         0.03370787 0.00449438]
 [0.         0.22921348 0.00224719 0.         0.00449438]
 [0.02696629 0.         0.10561798 0.00674157 0.01123596]
 [0.08314607 0.         0.         0.10786517 0.        ]
 [0.00674157 0.02921348 0.00898876 0.         0.13258427]]
```



- Decision Tree Classifier:

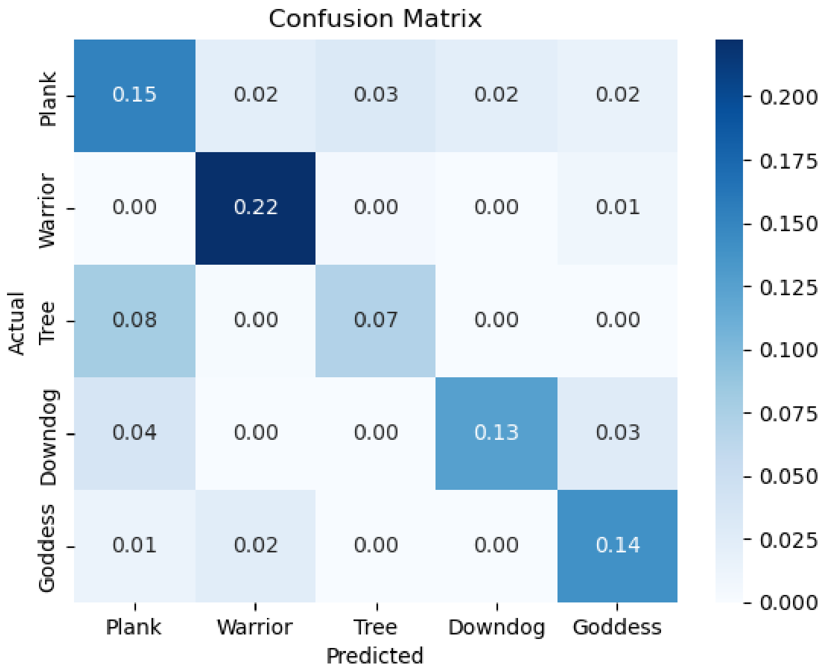
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score


# Define the hyperparameters and their values to search
param_grid = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search using cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Predict on the test set using the best model
predictions = best_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(Y_test, predictions)
print("Best Hyperparameters:", best_params)
print("Accuracy:", accuracy)

optmized_model_accuracy['DTC'] = accuracy
```

```
Best Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 10}
Accuracy: 0.7101123595505618
```

-

```python
# Calculate the raw confusion matrix
conf_matrix = metrics.confusion_matrix(Y_test, predictions, normalize='all')
print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.15280899 0.02247191 0.03146067 0.02247191 0.01573034]
 [0.         0.22247191 0.00449438 0.         0.00898876]
 [0.07865169 0.00224719 0.06966292 0.         0.        ]
 [0.03820225 0.         0.         0.1258427  0.02696629]
 [0.01348315 0.0247191  0.         0.         0.13932584]]
```

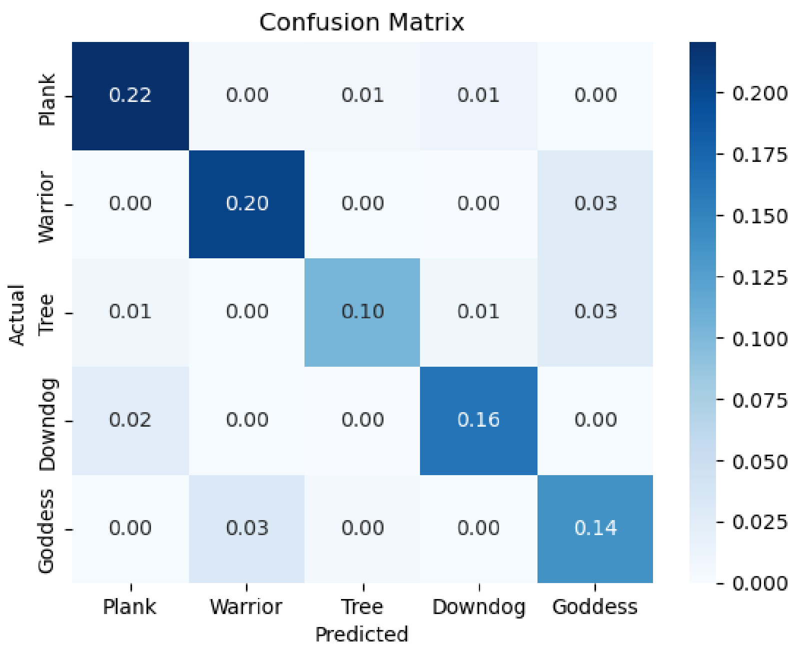


- K-Nearest Neighbor Classifier:

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Define a different set of hyperparameters and their values to search
param_grid = {
    'n_neighbors': [8, 10, 13, 14],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Perform grid search using cross-validation
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Predict on the test set using the best model
predictions = best_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(Y_test, predictions)
print("Best Hyperparameters:", best_params)
print("Accuracy:", accuracy)
```

```python
#conf_matrix = metrics.confusion_matrix(Y_test, predictions)
conf_matrix = metrics.confusion_matrix(Y_test, predictions, normalize='all')

print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.22022472 0.00449438 0.00674157 0.01123596 0.00224719]
 [0.00224719 0.20449438 0.00224719 0.         0.02696629]
 [0.00898876 0.00224719 0.10337079 0.00898876 0.02696629]
 [0.0247191  0.00224719 0.         0.16179775 0.00224719]
 [0.00224719 0.03146067 0.00449438 0.00224719 0.13707865]]
```

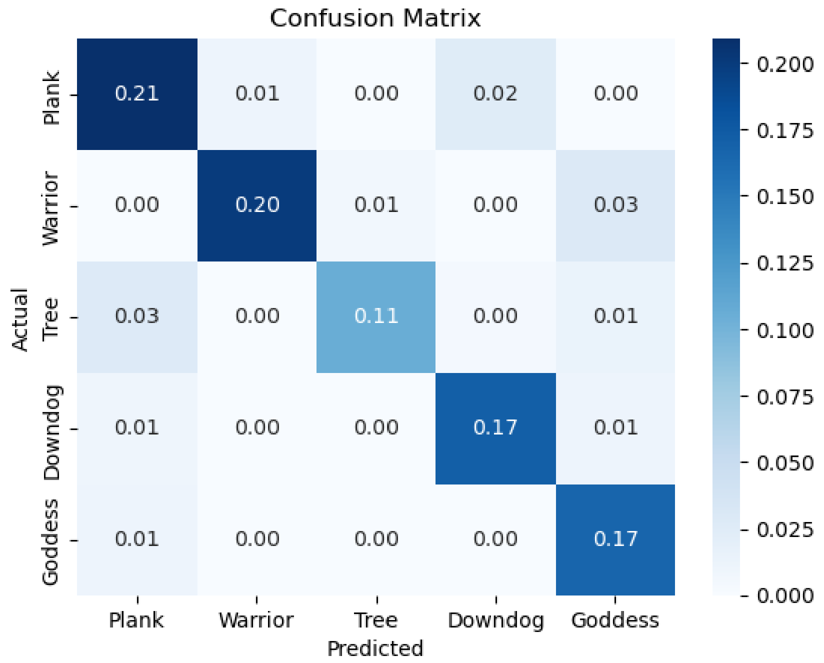


- Gradient Boost Classifier:

```
from sklearn.ensemble import GradientBoostingClassifier

# Create and train the Gradient Boosting Classifier
model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=42)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
#actual_labels = test_data['target']

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, predictions)
print("Accuracy:", accuracy)
optmized_model_accuracy['GBC'] = accuracy
```

```
conf_matrix = metrics.confusion_matrix(Y_test, predictions, normalize='all')

print("Confusion Matrix as Percentages:")
print(conf_matrix)

# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt=".2f", cmap='Blues',
            xticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'],
            yticklabels=['Plank', 'Warrior', 'Tree', 'Downdog','Goddess'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix as Percentages:
[[0.20898876 0.01123596 0.         0.0247191  0.        ]
 [0.         0.2        0.00674157 0.         0.02921348]
 [0.02696629 0.         0.10561798 0.00449438 0.01348315]
 [0.00898876 0.         0.         0.17078652 0.01123596]
 [0.01123596 0.         0.         0.         0.16629213]]
```



4.4 Model Evaluation:

```python
import matplotlib.pyplot as plt
import numpy as np

models = list(base_model_accuracy.keys())
base_accuracy = list(base_model_accuracy.values())
hyper_accuracy = list(optmized_model_accuracy.values())

# Generate positions for the models on the x-axis
x = np.arange(len(models))

plt.figure(figsize=(8, 6))

# Plotting the accuracy trend for base and hyperparameterized models
base_plot, = plt.plot(x, base_accuracy, marker='o', label='Base')
hyper_plot, = plt.plot(x, hyper_accuracy, marker='o', label='Hyperparameterized')

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Models')
plt.legend()
plt.grid(True)
plt.ylim(0, 1)  # Set the y-axis limits from 0 to 1 for accuracy scores
plt.xticks(x, models)  # Set x-axis ticks to model names
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability

# Annotate points with accuracy values
for i, acc in enumerate(base_accuracy):
    plt.text(x[i], acc, f'{acc:.2f}', ha='center', va='bottom', fontsize=10)

for i, acc in enumerate(hyper_accuracy):
    plt.text(x[i], acc, f'{acc:.2f}', ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()
```