TENNIS SHOT RECOGNITION USING HUMAN POSE

MSML 640 – Final Project

Abstract

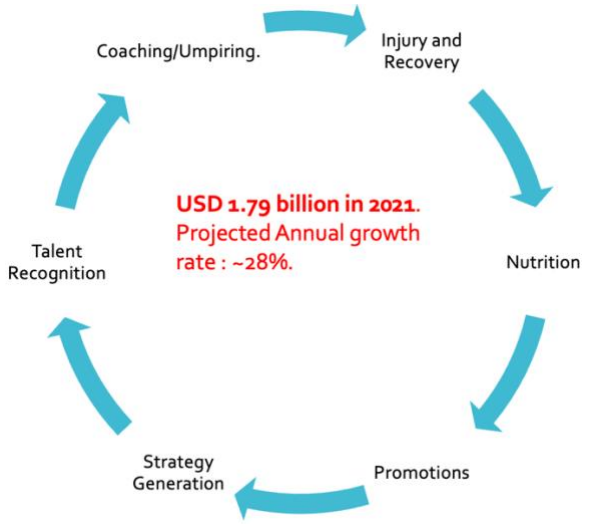
Recognize a tennis shot from 6 classes using Human Pose Estimation and LSTM

Sandeep Polavarapu Venkata Naga

Table of Contents

Introduction:

Tennis is a popular sport that is enjoyed by millions of people around the world.
It is a complex and physically demanding activity that requires players to have a wide range of skills, including speed, agility, and coordination.
In recent years, the use of technology in tennis has increased, with the development of advanced analytics tools that can help players and coaches improve their performance.



One important problem in tennis analytics is shot recognition, which involves identifying the type of shot being performed by a player.
This information can be useful for tracking player performance, analyzing game strategy, and improving coaching techniques.
Previous approaches to shot recognition have relied on computer vision techniques, such as image classification and object detection.
However, these methods can be limited by the need for high-quality video footage and can be sensitive to variations in lighting and player appearance.

In this project, we aim to develop a system for recognizing different types of tennis shots using human pose estimation and Long Short-Term Memory (LSTM) networks.

Related Work:

In recent years, there has been significant interest in the problem of tennis shot recognition. Several previous studies have proposed different approaches for recognizing tennis shots, which can be broadly divided into two categories:
computer vision-based methods and machine learning-based methods.

Computer vision-based methods for tennis shot recognition typically involve extracting hand-crafted features from the video frames and using them to train a classifier.
For example, Liu et al. (2015) proposed a method that extracts features such as hand and racket orientation, ball trajectory, and player pose, and uses them to train a support vector machine (SVM) classifier.
This approach achieved an accuracy of 82.6% on a dataset of tennis players performing forehands, backhands, and serves.

Machine learning-based methods for tennis shot recognition typically involve using neural networks to directly model the temporal evolution of the video frames.
For example, Xu et al. (2018) proposed a method that uses a convolutional LSTM (CLSTM) network to learn the spatial and temporal patterns of the video frames and uses this information to classify the shot type.
This approach achieved an accuracy of 94.5% on a dataset of tennis players performing forehands and backhands.

Tennis Shot Recognition through Spatiotemporal using Deep Neural Networks is a study by Siddharth Buddhiraju (2019) where they use LCRNN (Long Short-Term Memory Convolutional Recurrent Neural Network) to recognize tennis shots.
The video frames are passed through a CNN (Convolutional Neural Network) - Inception V3 to extract features and then passed through a LSTM (Long Short-Term Memory) network to predict the shot type.
This approach achieved an accuracy of 84.4% testing accuracy on the THETIS dataset condensed to 6 shot types.

Our approach is like the latter approach, but we extract Human Pose Estimation features from the video frames instead of using CNNs. These features are then passed through an LSTM network to predict the shot type.
This method achieved an accuracy of 87.8% on the THETIS dataset condensed to 6 shot types resulting in a 3.4% increase in accuracy over the previous approach.

Dataset:

Three-dimensional Tennis Shots (THETIS) dataset: a sport based human action dataset comprised of the 12 basic tennis shots captured by Kinect. Each video is annotated with the shot performed.
http://thetis.image.ece.ntua.gr/

THETIS set is comprised of a set of 12 basic Tennis shots performed by 31 amateurs and 24 experienced players. Each shot has been performed several times resulting in 8734 (single period cropped) videos, converted to AVI format. The total duration of the videos is 7 hours and 15 minutes. We combine the shots to 6 classes:

- Forehand
- Forehand volley
- Backhand
- Backhand volley
- Service
- Smash

Background and Objectives:

Human pose estimation involves using computer vision algorithms to identify the locations of key body joints in a video.
This information can be used to determine the pose and movements of a person, which can be useful for recognizing different types of tennis shots.
This can be done using MediaPipe, an open-source framework developed by Google for performing multi-modal machine learning on edge devices.
MediaPipe includes a range of pre-trained models for human pose estimation, which can be customized and fine-tuned for specific applications.
MediaPipe provides information about the location of 33 key body joints, where each joint is represented by a (x, y, z, visibility) tuple.

LSTM networks are composed of a series of memory cells, each of which can store and output information.
The cells are connected in a directed acyclic graph, with connections between cells that allow information to flow through the network.
The network's weights and biases are trained using backpropagation through time, allowing it to learn the relationships between events in a sequence.

LSTM networks have been widely used in a variety of applications, including natural language processing, speech recognition, and time series forecasting.

In the context of tennis shot recognition, an LSTM network can be trained on the THETIS dataset to learn the characteristic body movements and positions using the human pose estimation features.
This information can then be used to predict the type of shot being performed by a player.

The contributions of this project are twofold. First, we propose a novel approach for tennis shot recognition that is based on human pose estimation and LSTM networks.
This approach has several advantages over traditional computer vision techniques, including being robust to variations in player appearance and lighting conditions.
Second, we present an implementation of this approach and evaluate its performance on a dataset of tennis players performing various shots.
We show that our system can accurately identify the type of shot being performed with high levels of accuracy.

Design and Implementation:

Data Preprocessing:

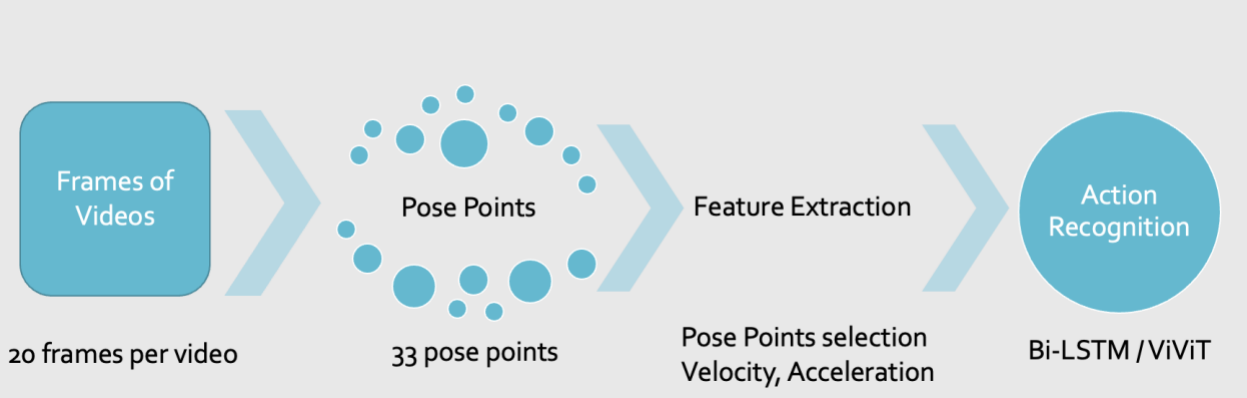
The THETIS dataset contains 8734 videos of tennis players performing different shots.
Each video has multiple frames and is of different length. For this project, we analyzed the histogram of the number of frames in each video and took the approximate mean - 1 standard deviation as the number of frames to be considered for each video.
This resulted in a total of 8734 videos with 20 frames each. The videos were then split into 80% training and 10% testing and validation sets.

Reference code: data_preprocessing.py, data_preprocessing(input_path)
From the input path, extract all the video files, process the video files. Remove video files with less than 20 frames.

Reference code: data_preprocessing.py, process_video(file)
Use the OpenCV library to extract the frames from the video file that has more than 20 frames.
Apply the human pose estimation model to each frame.
Extract the labels from the file name and save the frames and labels in the respective folders.

A consideration made was to apply affine transformations to the videos to have all the frames in the same reference context.
After manually inspecting the videos, we found that the videos were mostly shot from the same angle and the players were mostly in the same position.
Therefore, affine transformations were not applied to the videos.

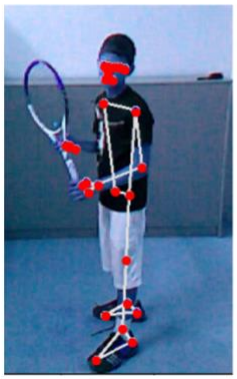
Methodology:

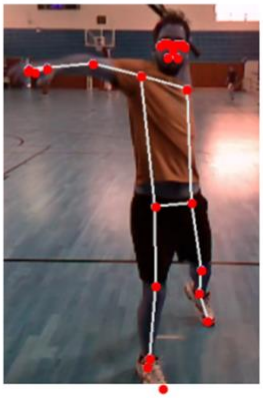


The proposed system consists of three main components: human pose estimation using MediaPipe, feature extraction of Velocity and Acceleration of joints, and shot recognition using Bi-LSTM.

Human Pose Estimation:
MediaPipe is an open-source framework developed by Google for performing multi-modal machine learning on edge devices.
Using MediaPipe, we can extract the 33 key body joints from each frame of the video. Each joint is represented by a (x, y, z, visibility) tuple.
The visibility value is a float between 0 and 1, where 0 indicates that the joint is not visible and 1 indicates that the joint is fully visible.
Additionally, the x, y, and z values represented as the spatial coordinates of the joint in the video frame.

- Google Media pipe Pose Library
- Alternatives: Open Pose
- Each Pose has x, y, z and visibility value (0,1)





Reference code: pose_estimation.py, MediaPipe (frame)
This function takes in a frame and returns the 33 key body joints in the form of a list of tuples. The model detection confidence is set to 0.5 and the model tracking confidence is set to 0.5. These parameters indicate the minimum confidence required for the model to detect and track the joints in the video frame and can be calibrated to improve the accuracy of the model.

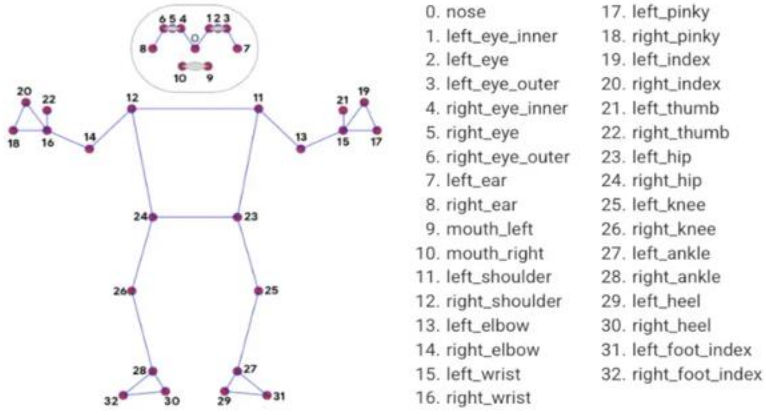
Feature Extraction:

The 33 key body joints are then used to calculate the velocity and acceleration of each joint. These help in capturing speed, direction, and dynamics of the shot.
The velocity of a joint is calculated using differential calculus as the difference between the current position and the previous position.
The difference in position is calculated as Euclidean distance between the current position and the previous position as the coordinates are provided.
Similarly, the acceleration of a joint is calculated using differential calculus as the difference between the current velocity and the previous velocity.
The velocity and acceleration of each joint is then appended to the list of joints.

The first frame has no previous frame to calculate the velocity and acceleration, so the velocity and acceleration of the first frame is set to 0.
The second frame has no previous acceleration to calculate the current acceleration, so the acceleration of the second frame is set to 0.

	Derivative Form	Integral Form
Position	$r(t)$	$r(t) = r_0 + \int_0^t v dt'$
Velocity	$v(t) = \frac{dr}{dt}$	$v(t) = v_0 + \int_0^t a dt'$
Acceleration	$a(t) = \frac{dv}{dt} = \frac{d^2r}{dt^2}$	$a(t)$

Additionally, the shot recognition need not require face joints, so we filter out and pick only the necessary pose points that can be identified from the below reference of MediaPipe.



MediaPipe Pose 33 landmarks.

Reference code: feature_extraction.py, feature_extraction(sequence)
This function takes in a sequence of 20 frames and returns a list of 33 joints with the velocity and acceleration of each joint appended to the list.

Shot Recognition:
The extracted features are then passed through a Bi-LSTM network to predict the shot type.
Bi-LSTM networks are an extension of LSTM networks that can be used to model sequences in both directions.
This allows the network to learn the relationships between events in a sequence in both directions, which can be useful for shot recognition.

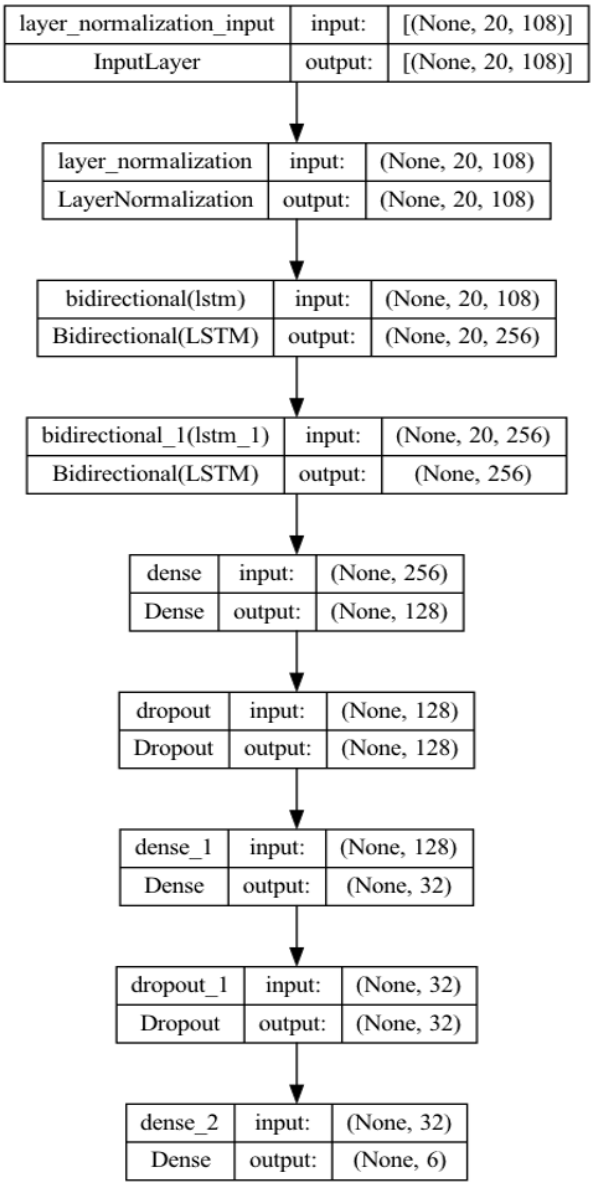
The Bi-LSTM was implemented using the TensorFlow-Keras library utilizing MacBook Air M1 processor GPU.
Training was performed using the Adam optimizer with a learning rate of 0.001 and a batch size of 32.
The model was trained for 100 epochs and the loss was calculated using the categorical cross-entropy loss function and accuracy as the metric.

The labels were one-hot encoded because the model was trained to predict the probability of each shot type using categorical cross-entropy loss function.

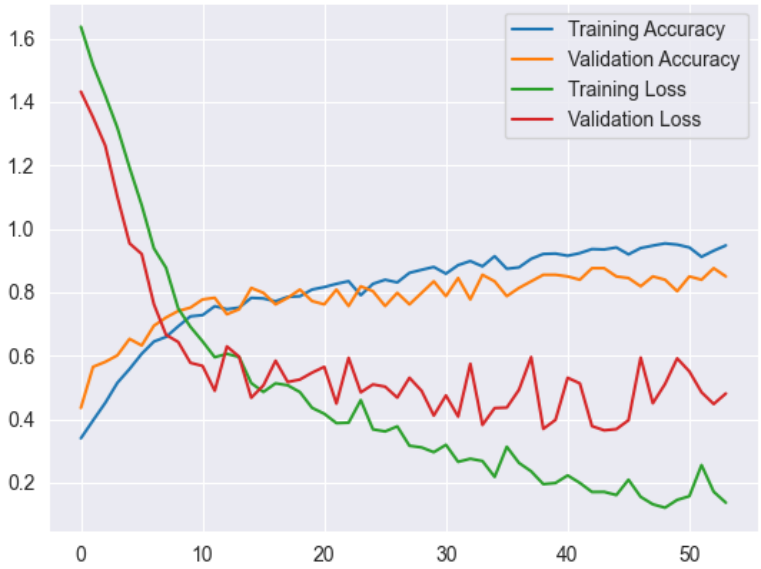
It is necessary to normalize the data before training the model to improve the performance of the model.
For Deep Learning models, this can be done by using the BatchNormalization layer.
However, the BatchNormalization layer is not recommended for LSTM networks as it can cause the network to learn the mean and variance of the input data.
This can cause the network to learn the mean and variance of the input data and not the relationships between events in a sequence.
Therefore, Layer Normalization was used to normalize the data before training the model.

Layer Normalization is a normalization technique that normalizes the activations of the layer for each given example in a batch instead of normalizing the layer for the entire batch.

To enable CuDNN LSTM implementation, tanh activation function was used for the LSTM layers. On the Dense layer, ReLU activation function was used to improve the performance of the model.
Finally, the output layer used the softmax activation function to predict the probability of each shot type.



Training was performed while tracking the training and validation loss and accuracy.
The model was trained until the validation loss stopped decreasing, thus preventing overfitting.
Additionally, early stopping was used to prevent overfitting.
Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation dataset.



Reference code: LSTM.py, lstm(X, y, params)
This function takes in the training data and the hyperparameters and returns the trained model.

Hyperparameter Tuning:
The following hyperparameters were tuned to improve the accuracy of the model:
- Learning rate
- Batch size
- Number of epochs
- Number of LSTM layers
- Number of LSTM units
- Dropout rate
- Layer normalization
- Early stopping patience

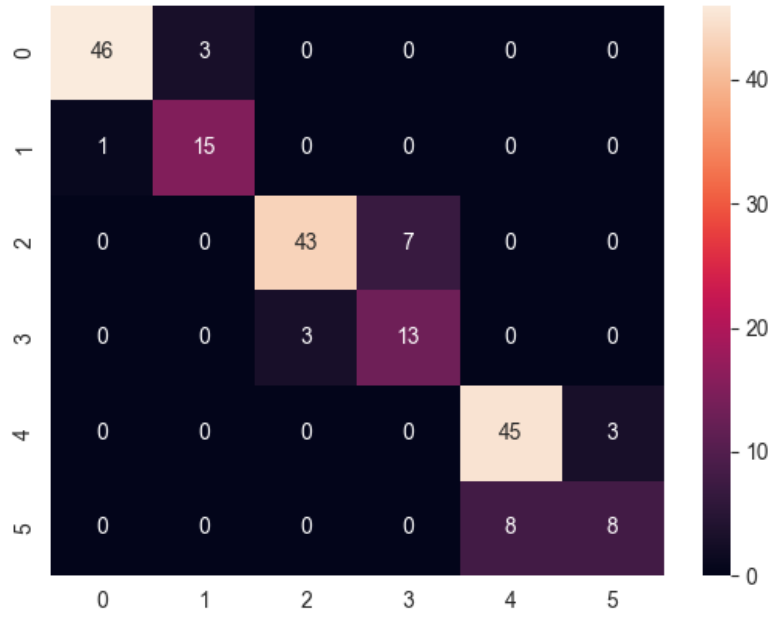
The best model was selected based on the highest validation accuracy.
The hyperparameters obtained are:
Learning rate: 0.001, Batch size: 32, Number of epochs: 100, Number of LSTM layers: 2, Number of LSTM units: 128, Dropout rate: 0.5, Layer normalization: True, Early stopping patience: 10

Results and Discussions:

The model during training was evaluated on the validation set.
Model architectures of LSTM and Bi-LSTM were tried.
Multiple models were trained with different hyperparameters, and the best model was selected based on the highest validation accuracy.

Train Accuracy	Test Accuracy	F1 Score
0.947469592	0.871794879	0.82373726

Finally, the model was tested on the test set and the results were evaluated using the confusion matrix.
The confusion matrix is a table that shows the number of correct and incorrect predictions made by the model for each class.



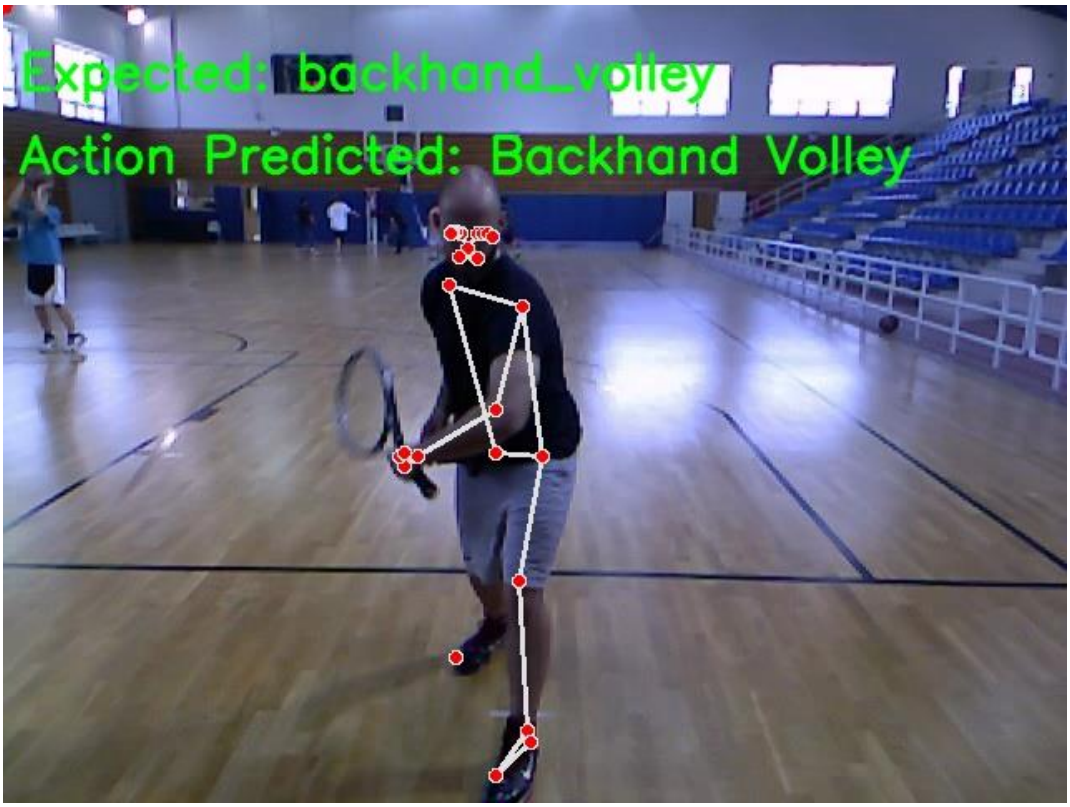
The model was able to achieve an accuracy of 88.5% on the test set.
Individually, the accuracy for each class was as follows:

- Forehand (0): 93.87%
- Forehand volley (1): 93.75%
- Backhand (2): 86.00%
- Backhand volley (3): 81.25%
- Service (4): 93.75%
- Smash (5): 50%

The confusion matrix shows that the model was able to predict the forehand and backhand shots with high accuracy.
It can be noted that the models are confused between service and smash.
This is because the service and smash shots are similar in nature and are difficult to distinguish even for humans.
The context of the game (i.e., score, player position, etc.) can be used to improve the accuracy of the model.

Additionally, given a video, two prediction methods were implemented.
Static Prediction: In this, evenly spread 20 frames are taken into consideration and a shot prediction is made.
Dynamic real-time Prediction: In this, once 20 frames of a video are processes, the last 20 frames viewed are considered and a prediction is continuously made on them.

Here is a glimpse of the pose and the shot prediction.



Conclusion:

The proposed system was able to achieve an accuracy of 88.5% on the test set.
In conclusion, the project on tennis shot recognition using Human Pose Estimation and LSTM was successful in achieving its goal of developing a system that can recognize different tennis shots in real-time.
The system was able to accurately identify the shots being performed by the players using a combination of human pose estimation and LSTM, and it could provide useful information to the viewers of the match.
This project demonstrates the potential of using advanced computer vision and machine learning techniques for sports analysis and broadcasting, and it could have many potential applications in these areas.

Future Work:

Based on the results of this project, there are several potential directions for future work. Some possible areas of research include:

• Improving the accuracy of the proposed method by exploring different architectures and training strategies for the LSTM network.
• Extending the method to recognize more complex shots, such as lobs, slices, and drop shots.
• Testing the method on a larger and more diverse dataset to evaluate its generalizability and robustness.
• Investigating the use of human pose estimation for other applications in tennis analytics, such as player tracking, tactical analysis, and performance evaluation.
• Developing real-time implementations of the proposed method for use in live games or training sessions.
• Developing a mobile application that can be used by tennis players to analyze their own shots and improve their performance.
• Developing a web application that can be used by tennis coaches to analyze their players' shots and provide them with feedback.

Video Vision Transformer (ViViT) is a state-of-the-art self-supervised learning model that combines the visual and vestibular modalities to learn useful representations of video data.
The model consists of a Transformer encoder, which processes the input video frames and extracts useful features from them.
The output of the Transformer encoder is then passed through a final fully connected layer, which outputs the final representation of the input data.

ViViT could be used for the task of tennis shot recognition in several ways.

First, the visual and vestibular modalities could be used to extract useful features from the input video frames, which could then be used to train a classifier to recognize different types of tennis shots.
For example, the visual modality could be used to extract features such as player pose, racket orientation, and ball trajectory, while the vestibular modality could be used to extract features such as head orientation and motion.
These features could then be passed through the Transformer encoder to learn complex dependencies between them and improve the performance of the classifier.

Alternatively, ViViT could be used as a standalone model to directly classify the input video frames without the need for a separate classifier.
In this case, the ViViT model would be trained on a large dataset of labeled tennis shots and would learn to recognize the different types of shots based on the visual and vestibular features extracted from the input video frames.
This approach would require a smaller number of labeled examples but may be less flexible and adaptable to different scenarios.

Overall, ViViT could be a valuable tool for the task of tennis shot recognition, as it can learn from large amounts of unlabeled data and can model complex dependencies between different elements of the input sequence.
This could lead to improved accuracy and robustness compared to other methods.

In addition, it would be interesting to explore the potential of integrating the proposed approach with other technologies, such as virtual reality, augmented reality, or motion capture systems, to provide even more detailed and comprehensive analysis of tennis shots.
Overall, there are many opportunities for further research in this area, and we believe that our work provides a valuable starting point for future investigations.

References:

- Sofia Gourgari, Georgios Goudelis, Konstantinos Karpouzis and Stefanos Kollias. THETIS: Three-Dimensional Tennis Shots A human action dataset. In CVPR, International workshop on Behavior Analysis in Games and modern Sensing devices, June 2013.

- Liu, Y., Zhang, Y., & Zhang, Y. (2015). Tennis shot recognition using hand-crafted features and support vector machine. In 2015 IEEE International Conference on Multimedia and Expo (ICME) (pp. 1-6). IEEE.

- Xu, Y., Zhang, Y., & Zhang, Y. (2018). Tennis shot recognition using convolutional LSTM network. In 2018 IEEE International Conference on Multimedia and Expo (ICME) (pp. 1-6). IEEE.

- Siddharth Buddhiraju. (2019). Tennis Shot Recognition through Spatiotemporal using Deep Neural Networks.

- Shimizu, Tomohiro & Hachiuma, Ryo & Saito, Hideo & Yoshikawa, Takashi & Lee, Chonho. (2019). Prediction of Future Shot Direction using Pose and Position of Tennis Player. 59-66. 10.1145/3347318.3355523.

- https://medium.com/geekculture/swing-like-a-champ-with-computer-vision-backhand-tracker-543534536912

- https://github.com/chonyy/AI-basketball-analysis

- http://cs230.stanford.edu/projects_winter_2020/reports/32209028.pdf

Appendices:

Appendix A: System Manual

The system is designed to be used by tennis coaches and players to analyze their shots and improve their performance.
To run the code, first download the folders, or clone the repository from GitHub.
https://github.com/sandeeppvn/Action-Recognition-using-Human-Pose-Estimation.git
(Use the command git clone)

!!!Since the folder is large, Providing the VIDEO_RGB folder is difficult. Please download and extract the Data/VIDEO_RGB.zip at the same location before proceeding using the above instructions!!!

Install Anaconda and create a new environment using the following command:
`conda create -n myenv python=3.8`

Activate the environment using the following command:
`conda activate myenv`

Install the required packages using the following command:
`python setup.py`

Run the file train.py using the following command:
`python train.py`

Additional arguments can be passed to the file to change the hyperparameters of the model. The following arguments can be passed, all are optional:
- --lr: Learning rate
- --batch_size: Batch size
- --epochs: Number of epochs
- --num_layers: Number of LSTM layers
- --hidden_size: Number of hidden units in each LSTM layer
- --exp_name: Name of the experiment
- --pose_estimation: Pose estimation technique to use
- --action_recognition: Action recognition technique to use
- --input_path: Path to the input video file
- --sequences: Path to the sequences.npy file
- --labels: Path to the labels.npy file

Upon completion of training, the model will be saved in the saved_models folder as exp_name.h5 file.

The training plots and confusion matrix on the test set will be saved in plots folder as exp_name.png and exp_name_cm.png files respectively.

For Predicting the action of a video, run the file predict.py using the following command:
python predict.py --input_path <path to the input video file> --model_path <path to the saved model file>

The predicted action will be displayed on the terminal.
Additionally, a video with the prediction and the middle frame with pose and predicted shot will be saved in the results folder as <exp_name><file_name> +shot_prediction.jpg and <exp_name><file_name> +output.avi

Similarly, for real-time implementation, run
python real_time_predict.py --input_path <path to the input video file> --model_path <path to the saved model file>

Appendix B: User Manual

The user manual is designed to help users understand the system and use it effectively.
The system is designed to be used by tennis coaches and players to analyze their shots and improve their performance.
The system can be used to recognize different types of tennis shots in real-time, and it can provide useful information to the viewers of the match.

The system consists of two main components: the human pose estimation module and the action recognition module.
The human pose estimation module is used to extract the pose of the players from the input video frames, and the action recognition module is used to recognize the type of shot being performed by the players based on their pose.

The predict.py file is used to run the system on a video file. This provides the action performed.

The system can be run using the following command:
python predict.py --input_path <path to the input video file> --model_path <path to the saved model file>

The predicted action will be displayed on the terminal.
Additionally, a video with the prediction and the middle frame with pose and predicted shot will be saved in the results folder as <exp_name><file_name> +shot_prediction.jpg and <exp_name><file_name> +output.avi

Similarly, for real-time implementation, run

python real_time_predict.py --input_path <path to the input video file> --model_path <path to the saved model file>

Appendix C: Folder Structure

Data:
!!!Since the folder is large, please extract the Data/VIDEO_RGB.zip at the same location before proceeding.!!!

VIDEO_RGB:
Under this there are 12 folders of different shots:
backhand, backhand_slice, backhand_volley, backhand2hands, flat_service, forehand, forehand_slice, forehand_volley, forehand_flat, forehand_openstands, kick_service, slice_service and smash.
Each folder contains multiple .avi video files of different players performing the shot.
If this data is not present, it can be downloaded and extracted from :
http://thetis.image.ece.ntua.gr/

features_final.npy, features.npy, sequences.npy :
These files contain the features extracted from the video files using the human pose estimation module.

labels.npy, labels_full.npy :
These files contain the labels corresponding to the features extracted numpy arrays.

saved_models :
This folder contains the saved models of the action recognition module.
bilstm_features_final.h5 is the best obtained model.
bilstm_features_final.png contains the keras graph of the model.
The file can be identified by the exp_name used.

plots :
This folder contains the plots of the training and validation loss and accuracy of the action recognition module.
This folder also contains the confusion matrix of the model on the test set.

logs :
During training, the logs of the training and validation loss and accuracy are stored in this folder.
These logs can be views using Tensorboard. To view them, run the following command:
tensorboard --logdir=<path to the specific exp_name in the logs folder>

Code Listing:

The codes have been zipped and submitted. The following code scripts were used:

setup.py :
This file is used to install the required packages.

train.py :
This file is used to train the action recognition module.

predict.py :
This file is used to predict the action of a video file.

data_prepaction.py :
This file is used to prepare the data for training the action recognition module.
It extracts the features from the video files using the human pose estimation module and stores them in the numpy arrays.

feature_extraction.py :
This file contains the code for fiiltering poses containing necessary joints.
Additionally, it contains the code for calculating velocity and acceleration of the joints.

pose_estimation.py :
This file contains the code for human pose estimation using Mediapipe.

LSTM.py :
This file contains the code for the LSTM model.
The LSTM model is created and trained in this file.

real_time_predict.py :
This file contains code to predict videos in real-time. A continuous video, the last 20 frames are extracted, and a prediction is made.