Predictive Pre-Training of Tennis Shot Embeddings

E6691.2024Spring.TECO.report.as7092.gyt2107.fps2116
Tawab Safi as7092, George Tamer gyt2107, Flor Sanders fps2116
Columbia University

***Abstract*—AI sports coaching promises to enhance athletic performance in sports like tennis. This paper presents the development of a deep learning framework that learns useful representations of tennis shots from broadcast videos. We formulate this as a two step problem - first, reconstructing the 3D workspace of the tennis environment from a monocular video feed, and second, modeling motion from this 3D information. Our workspace construction solution uses a combination of classical and deep learning based computer vision, while our deep learning motion model leverages graph neural networks and sequence modeling. We find that while our system struggles to predict future motion of a player, its learned representations are immediately useful for downstream tasks such as hit classification, showing this is an exciting avenue for future research.**

1. Introduction

Sports and athletics are an integral part of recreation in modern society. There are numerous benefits to playing sports, including physical fitness and social connection. However, many sports, such as tennis and golf, are difficult to learn, posing a high barrier to entry before they become enjoyable and good workouts. Sports like tennis can be broken down into shot technique and strategy. Consequently, a lot can be learned from the movement of players to hit one shot, and the decisions they make to shape a point. Coaching is the best way to improve at such sports, but human coaches are prohibitively expensive for most people, and scheduling is inconvenient compared to digital solutions. For this reason, AI sports coaching is an avenue of research with high societal payoff.

We posit that the key task to enable further research in coaching is the ability to learn useful representations of a single sports shot. For example, a single-shot representation could be used as input for a multi-modal system to describe shot technique in natural language, or composed into a sequence to represent a point, to learn about strategy. Furthermore, we posit that at its core this is less of a computer vision challenge and more of a robot learning challenge - we care less about visual details than about the motion of a player, and how different shot motions lead to different outcomes. These two hypotheses anchor the rest of this paper - we aim to build a model that learns useful representations of a tennis shot from a player's local and global movement.

We formalize the problem: given a monocular video feed of a professional tennis match, our task is to first reconstruct the environment's 3D-workspace, and then is to learn representations of player movement in that workspace.

Deep learning in sports has been applied to many areas of sports across perception, comprehension, and decision making, with work on learning a task from video data or from graphs [1]. However, few graph-based works discuss this problem end-to-end. We believe that data extraction from video to workspace should be considered jointly with learning in the workspace.

To that end, we introduce an end-to-end data pipeline for workspace construction from sports broadcast videos, and a deep learning model designed to learn motion from the reconstructed workspace. Here, our pipeline and model are built for tennis, but each component could be expanded to other sports in the future. Our workspace reconstruction pipeline uses projective geometry, homography, and 3D-pose reconstruction to recover local and global 3D position information of both players in a tennis match. Our model uses graph neural networks and sequence modeling to learn representations of a player's motion.

The rest of this report is structured as follows. Section 2 gives an overview of previous works related to the deep learning tasks employed in the current project. In Sections 3 and 4 the methodology of this project is outlined and its implementation is explained in more detail. The results are presented in Section 5. Finally, Sections 6 and 7 respectively draw conclusions and provide an outlook on future research directions.

2. Related Work

Following from our problem formulation, we present the related work in two parts: workspace reconstruction and motion learning.

2.1. Workspace Reconstruction

2.1.1. Pinhole Camera Model and Homography.

Modern cameras follow the pinhole camera model, in which we can describe the 3D to 2D mapping as a projective transformation [2]. Homography is a sub-field of classical computer vision relating to linear transformations between different views of an image lying on the same plane, and corresponding homography matrices can be used in combination with known object dimensions to recover camera calibration [3].

2.1.2. 3D-pose reconstruction.

Recovering 3D information from video is a vibrant field of research. One line of work focuses on generic point tracking within video, optionally trying to recover 3D information [4], [5]. Another line of work focuses on extracting human or animal poses with predefined keypoints [6], [7]. We focus on pose-specific models in this paper, given we do not need generic point mapping.

2.2. Motion Learning

2.2.1. Geometric Deep Learning.

Neural networks that can learn geometric priors are applicable to a large variety of fields, and graph neural networks are a popular choice for representing geometric data [8].

2.2.2. Sequence Modeling.

Lastly, sequence modeling is one of the most universal paradigms in deep learning, enabling learning across many modalities as diverse as language, video, audio, genomics, and time-series.

Recurrent neural networks (RNNs) are a category of sequence models that maintain a fixed-size hidden state, calculated at each step of the sequence, to represent an inputted sequence [9]. There are many variants of RNNs including the Long Short-term Memory (LSTM) architecture and the Gated Recurrent Unit (GRU) [10], [11].

Another family of sequence models is the transformer. Transformers are categorized by their attention operation, which considers the relationships between every pair of inputs in a sequence, and learns how much to attend to each one [12]. They are more computationally expensive but more powerful than RNNs. Given the physics prior of a tennis shot, we believe transformers were not needed for our task.

3. Methodology

In this section we describe the project at hand from a birds-eye perspective. The goal is to develop a deep learning architecture that generates semantically meaningful embeddings of tennis players performing shots. To achieve this goal, the system architecture shown in Figure 1 is used.

The key steps in our methodology are as follows.

1) Develop an automatic video frame annotation system, which extracts useful features from video footage of a tennis shot. Thus, the data dimensionality of the feature space shall be drastically reduced.
2) Design and implement a deep learning architecture that extracts embedding vectors of fixed size from the video frame annotations.
3) Pre-train the model on a self-supervised task to learn semantically meaningful embeddings, based on the training data.
4) Fine-tune the model on a downstream task, taking the tennis shot embedding as input features for the downstream model.

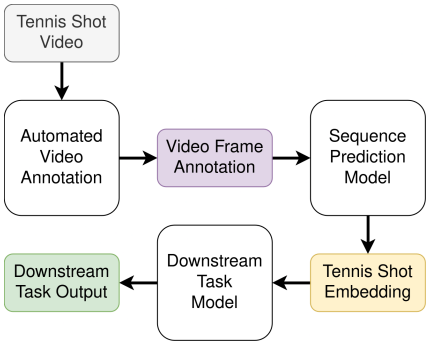


Figure 1: System architecture.

This project uses 3D pose prediction of the next frame as a pre-training task and evaluates the model on the hit classification downstream task. As the name suggests, the objective is to classify the type of response hit (flat/topspin/slice/unsure and forehand/backhand). Though quite elementary, this task serve as a proof of concept validity and open up the road to more complex downstream tasks like player coaching in conjunction with large language models.

4. Implementation

In what follows, we provide more detail regarding the implementation specifics for the methodology outlined in Section 3.

4.1. Dataset

In this project, we make use of the Tenniset dataset [13]. It contains video material of five full-length tennis matches, labeled V006 to V010, played at the 2012 Olympic Games in London. The resolution of the videos are 1280×720 pixels with a frame rate of 25 frames per second. In terms of annotations, it provides frame indices for start and end times of temporal events such as serves, hits, points, sets and games. Additionally, the tool used for data annotation is equally published under and open source license, which facilitates the future labeling of inference samples [14].

A key element of this project is the further annotation of this dataset with player poses and positions on the tennis

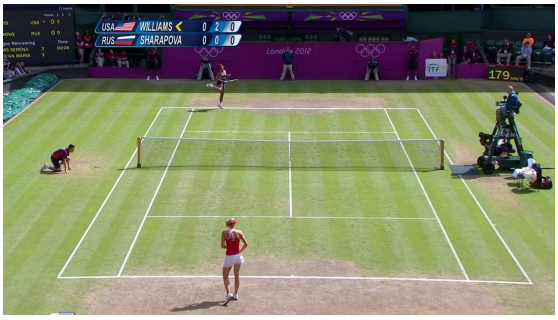


Figure 2: Sample Tenniset video frame.

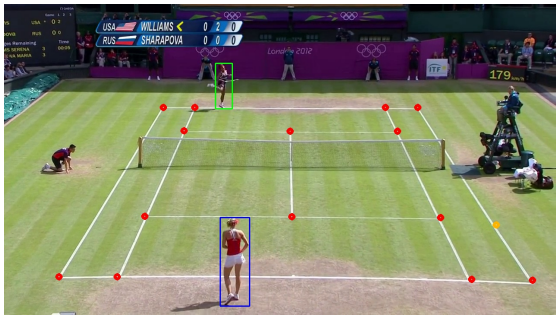


Figure 3: Sample annotated Tenniset video frame. Player bounding boxes are shown in blue (near) and green (far), court points in red and the tennis ball in yellow.

field, as well as ball trajectories. However, accurate annotation of features in video data is an arduous process and this feature extraction would have to be performed at inference time as well. For these reasons, our objective is to establish a pipeline that automates the desired feature extraction through utilization of existing deep learning models, making use only of the existing annotations included in the Tenniset data.

The feature extraction is implemented in the following five steps, each of which will be explained in more detail in the paragraphs below.

1) Splitting of tennis matches into shot segments.
2) Performing ball, court and player bounding box detection.
3) 2D player pose detection, bounding box refinement and global player position extraction.
4) 3D player pose detection.
5) Training, validation and test splitting.

The final processed dataset is available for download on Google Drive.

4.1.1. Tennis Match Splitting.

The goal of our model is to learn useful latent representations of tennis shots for use in various downstream applications. As such, we desire to split the tennis match videos into segments containing only a single shot. This is easily achieved by merging the serves and hits annotations of the Tenniset dataset and trimming the video into fragments based on their starting and ending frames.

At this point, we additionally extract some auxiliary data, such as which player is performing the shot, whether it is a serve or a return hit and the class labels for the downstream tasks. Finally, we annotate each sequence with a validity flag, such that samples of poor quality can be filtered before training.

4.1.2. Ball, Court and Player Detection.

One could expect the detection of the tennis court, the player bounding boxes and the tennis ball to be the most challenging task in the data annotation pipeline. Lucky for us, tennis enthusiasts that came before us have spent significant time and effort on techniques to extract these features from video material.

In this project we make use of the TennisProject implementation [15]. This implementation makes use of deep learning networks for all three aforementioned tasks, in contrast to older alternatives like Tennis Tracking [16] which makes use of less robust classical machine vision for tennis court detection. Specifically, TennisProject makes use of the following models for each task:

- **Ball Detection**: TrackNet, a model which is especially designed for detecting small and fast-moving objects by processing consecutive frames simultaneously [17], [18].
- **Court Detection**: A modified version of the TrackNet architecture is used with a single frame input and individual output layers for each of the 14 detected court points. After detection, post-processing is applied to the court points using classical machine vision and homography w.r.t. a reference image to improve the quality of the detected points [19].
- **Player Detection**: Faster R-CNN, pre-trained on the COCO dataset to detect people on either half of the court [20], [21].

While this collection of models works quite well out of the box, as illustrated by Figure 3, some issues arise when applying them in practice. First, for video fragments taken from an angle where the court lines are not visible, the court detection model naturally fails to predict meaningful points of interest. In such cases, we mark the problematic frames as invalid and remove them from further processing steps. Another issue arises when people other than the player find themselves near the tennis court (e.g. the ball attendant, line judge or a coach), in which case Faster R-CNN detects multiple people. This issue is resolved by making use of a heuristic, where we assume that the player is the person who is closest to the center of the court end line.

By means of sample visualization, we evaluate the quality of our data labeling thus far. In the majority of cases, the result are close to perfect with well-defined court points and correct player bounding boxes. The tracking of the tennis ball proved unreliable in some cases where the ball was moving at very high speeds or when out of the camera's sight, resulting in quite a few missing or mislabeled points.

Furthermore, Faster R-CNN is not perfectly accurate and as it does not take the temporal relation between subsequent samples into account, the player is misidentified in some cases.

4.1.3. Bounding Box Refinement, 2D Pose Estimation and Global Player Position Extraction.

It is by now well-known that the quality of the data set on which a deep learning model is trained has a significant influence on its final performance [22]. Therefore, one of the main goals of this processing step is to refine the player bounding box predictions to resolve the mistakes discussed in the last section.

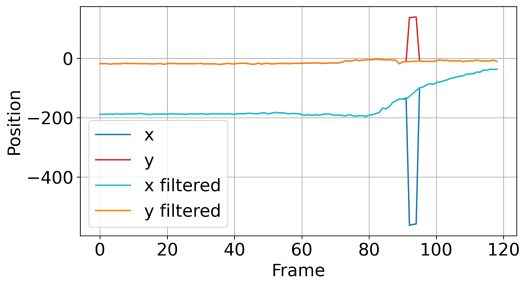


Figure 4: Player bounding box interpolation.

To achieve this goal, we adopt the following approach. First, a temporal analysis is performed on the existing bounding box predictions to identify missing points and possible mistakes, e.g. when quick changes in a player's center position or the bounding box area occur. An example is shown in Figure 4. For each frame and each player, we propose a number of candidate bounding boxes based on the data available to us. If the frame is determined to be likely valid, the existing bounding box is used as the only candidate; If the frame is likely a mistake, we use the following three candidates.

- Player bounding box of the last frame.
- Linear interpolation between likely valid bounding boxes, as determined by the temporal analysis.
- Existing bounding box prediction.

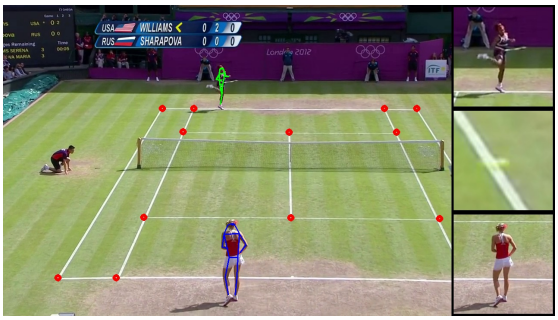


Figure 5: Sample annotated Tenniset video frame. Player 2D poses are shown in blue (near) and green (far) and the court points in red.

The frame is cropped to the bounding box candidate with 50 pixels of padding on all sides and is processed using the "human" pose estimation model from the MMPose framework [23]. In the background this makes use of the RTMDet-m model to predict a new set of bounding boxes and of the RTMPose-m model to simultaneously extract 2D pose information [24], [25]. In the rare case where multiple people are detected in the cropped frame, the one closest to the center is assumed to be the player. A sample result is given in Figure 5.

This new set of extracted bounding boxes show significantly improved temporal consistency compared to our previous predictions, though visualizations show that some mistakes are still unresolved. The final bounding box predictions, in conjunction with the tennis court predictions and their homography matrices, are employed to anticipate the global X and Y positions of each player on the tennis court. These positions are normalized relative to the court's width and height, with the origin situated at the court's center.

4.1.4. 3D Pose Detection.

The original idea for the next step was to perform classical machine vision techniques to reconstruct 3D information for the extracted points of interest in the frame. It turns out this is significantly more challenging than expected, thus we turn to direct 3D player pose estimation. This means that we, unfortunately, do not further consider the ball trajectory information.

We use the open-source MotionBert model to perform 3D pose detection [6]. MotionBert is a motion encoder that takes video and 2D skeleton annotations as input and outputs 3D pose estimation across time. The core of MotionBert is an encoder that was pre-trained in a self-supervised manner to predict 3D motion from corrupted 2D skeletons, from a datset of video and motion capture data. Then, a simple MLP is added and the model is fine-tuned for the 3D pose prediction task. MotionBert uses the HM36 keypoint format derived from the Human 3.6M Dataset, and we maintain this format [26], [27].

To make MotionBert predictions useful for us, we employ two post-processing techniques: scaling and rotation.

First, we note that the X,Y,Z coordinates outputted by the model are much smaller than a typical human. To solve this, we scale each 3D skeleton such that the detected torso corresponds in length to 105cm.

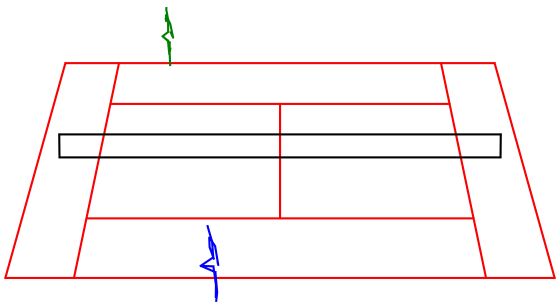


Figure 6: Sample Tenniset frame 3D annotation.

Second, we note that the poses detected by the model are not upright in orientation, nor does their angle of rotation correspond to the camera view or any other consistent phenomenon. In order to resolve this, we match key points (shoulders, arms, hips and legs) between the detected 2D and 3D poses and derive the rotation matrix that transforms as closely to the 2D pose orientation as possible [28]. A visualization of final pose samples for the frame of Figure 2 are given in Figure 6.

4.1.5. Graph Construction.

We then convert each 3D pose into a graph with the following algorithm:

Algorithm 1 Graph Construction Algorithm

1: Input:
    1. POSE = (X, Y, Z), where X, Y, and Z are each of dimensions 1 by 17, the number of nodes in the HM36 pose format
    2. SKELETON = (JOINT START, JOINT END), where
2: OUTPUT:
    1. A single vector [1 x output-dim], which represents our shot representation
3: ALGORITHM:
    1. Initialize V := [], E := []
    2. Add vertices: for each joint j made from joint = (X[j], Y[j], Z[j]), V ¡- V + joint
    3. Add edges: for each b in SKELETON, bone = (SKELETON[b][0], SKELETON[b][1], E ¡- E + bone

4.1.6. Training, Validation and Test Splitting.

In our approach to splitting the data for training, validation, and testing, we strategically designate video V010 exclusively for testing to avoid data leakage between training and testing stages. For the training and validation datasets, 80% of the shot sequences from each video are allocated to the training set, and the remaining 20% to the validation set. Post-distribution, we conduct a data integrity check to filter out sequences that either lack pose information or are incorrectly formatted. This validation step results in a minimal reduction of the available data, retaining over 98.5% of the data across training, validation, and test sets. The final counts for the datasets are as follows: 2,248 shot sequences for training, 565 for validation, and 370 for testing.

4.2. Deep Learning Network

The architecture of the deep learning network is presented in Figure 7. At a high level, the model operates as follows.

1) For each frame, the features are fed into the frame embedder module which encodes the information into a frame token vector of fixed length d_{fe}.
2) The sequence of frame tokens is subsequently passed through a recurrent neural network (RNN) sequence model.
3) The output of the RNN is a sequence of frame embeddings, the last of which is interpreted as an embedding of the complete tennis shot.

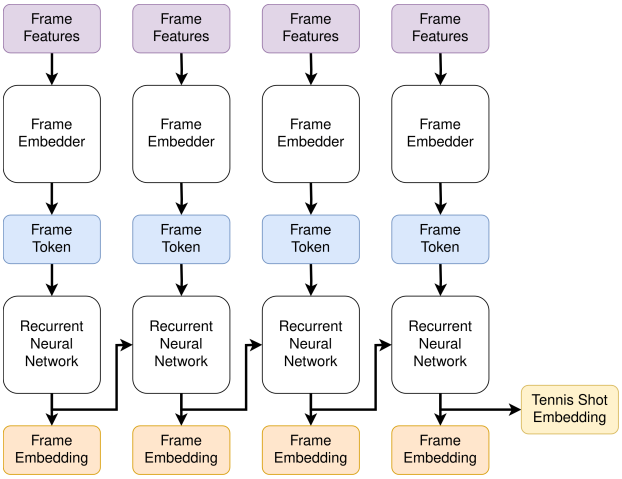


Figure 7: Model architecture.

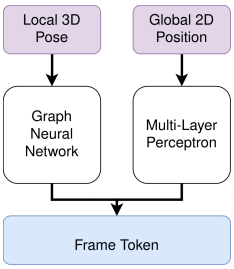


Figure 8: Model frame embedder sub-network.

Figure 8 presents the working of the frame embedder in more detail. The input features of the embedder are the local 3D pose information of the active player (i.e. the player performing the shot), along with their global 2D position on the court. Respectively, the player pose is encoded into a graph structure and fed through a graph neural network (GNN) architecture, while the global 2D position is encoded using a multi-layer perceptron (MLP).

The paragraphs below discuss the three sub-networks present in the network in more detail.

4.2.1. Graph Neural Network.

In the first stage of our model, we employ a graph neural network (GNN) to learn a representation of frame features. Here, we leverage a Graph Attention Network (GAT), which applies many layers of Bahdanau attention on graph nodes to learn node and graph level representations [29], [30]. We implement the GAT with two hidden layers and a hidden dimension of 32. Compared with Graph Convolution Networks (GCNs) [31], GATs allow our network to implicitly assign different importance to neighbors within the neighborhood of a node. We posit that this is preferable in a sports domain. For example, the shoulder and elbow of

the arm holding the racket should be more important than the shoulder and elbow of the arm not doing so, and these are both in the neighborhood of the upper torso node.

4.2.2. Positional Embedding.

A global positional embedding is generated by passing the global 2D player coordinates on the tennis court through a dense MLP network. We use one hidden layer with a hidden dimension of 32. The output is a vector of the same dimension as the GNN output and both vectors are added together to generate the frame token.

4.2.3. Recurrent Neural Network.

As a sequence model, we make use of a unidirectional stacked RNN architecture. More specifically, the RNN implementation of choice is the long short-term memory (LSTM) network. It has two hidden layers and a hidden dimension of 64.

4.3. Model Pre-Training

Inspired by the success in language models [32]–[34], we pre-train our network on a self-supervised learning task. For pre-training, the sequence of frame embeddings produced by the deep learning model (cfr. Figure 7) is used as an input to a secondary network that is tasked with predicting the 3D pose information contained in the features of the next frame, as visualized in Figure 9.

The complete network is jointly trained. As such, the RNN should learn parameters that encodes the movements of players over time while performing tennis shots. At the same time, the GNN and MLP backbone elements should learn meaningful representations that helps the RNN achieve its task.

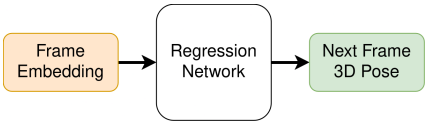


Figure 9: Model pre-training task.

In practice, we prepare our data by loading the training, validation and test datasets into their respective data loaders. In here, the 3D pose and 2D position data are respectively normalized such that the length of a person's torso always measures 1 and the tennis court ranges from -1 to 1 along both axes. Furthermore, the data is divided into batches of fixed size and the sequences of each batch are padded to the maximum batch sequence length. Finally, a mask is implemented to ensure that the padded regions do not influence loss computation and the backward pass. The model is pre-trained for 100 epochs on a Nvidia T4 GPU with a learning rate of 10^{-3} and a batch size of 32, with the Adam optimizer used to minimize the mean squared error (MSE) loss function. The training history is visualized in Figure 10.

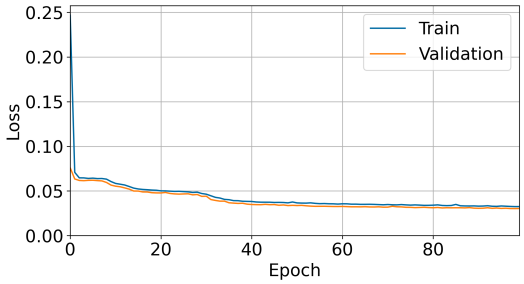


Figure 10: Pre-training history.

4.4. Downstream Task Fine-Tuning

To evaluate the effectiveness of the learned shot representations from the pre-trained model, we fine-tune the model on the hit classification downstream task. The Tenniset dataset [13] provides 11 labels for hit classification, as shown in Figure 12. However, due to the uneven distribution of labels in the dataset and the incomplete information provided by certain labels like "Backhand_Unsure" and "Forehand_Unsure," we focus only on the top four labels with the highest frequency for the downstream hit classification task. The distribution of these top four labels in the training dataset is presented in Figure 13. A similar distribution across the labels is observed in the validation and test dataset.

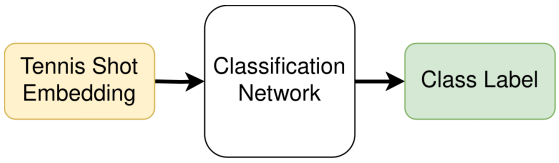


Figure 11: Downstream Task Architecture

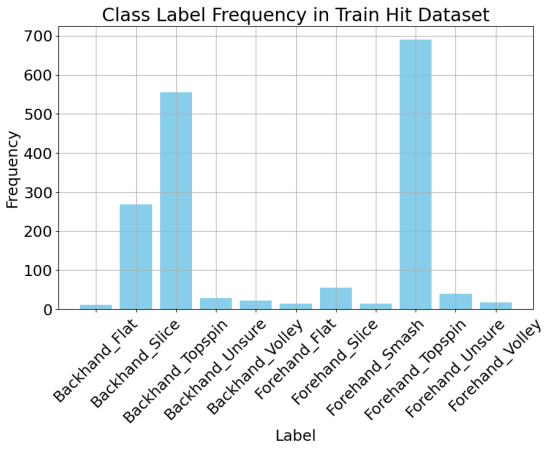


Figure 12: Complete Label Distribution in Training Dataset

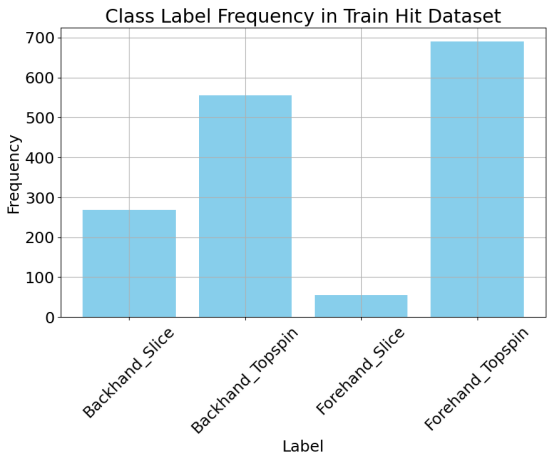


Figure 13: Top 4 Label Distribution in Training Dataset

The fine-tuning procedure involves replacing the output head of the pre-trained model with an MLP-based classification network and training it on the hit classification downstream task. The classification network has two hidden layers, with hidden dimension of 512, followed by an output layer. The diagram of the downstream task architecture is shown in Figure 11.

4.4.1. Training Procedure.

The training procedure for fine-tuning involves two main phases:

1) **Training the Classification Network**
   We replace the output head of the pre-trained model with a new classification network that takes the last frame representation as the shot representation and uses it to predict one of the four labels. During this phase:
   - All layers of the pre-trained model are frozen.
   - Only the new classification network is trained for 50 epochs.
   - The objective is to have the classification network learn to best use the shot representations from the pre-trained model for hit classification.
2) **Phase 2: Fine-Tuning the Entire Model**
   In this phase, we unfreeze all layers of the pre-trained model and train the entire model for an additional 50 epochs.
   - The classification head is initialized with weights from Phase 1, which has effectively learned how to use the pre-trained shot representations.
   - The complete model is trained to adjust the shot representations specifically for the downstream task. This should enable the model to achieve higher accuracy on our specific downstream classification task.

4.5. Software Design

The complete implementation for this project is available on Github. The src directory of the repository is organized into four main parts, each contained in its own sub-directory.

- $data$: The automated video annotation, described in Section 4.1 is implemented here. Specifically, the code for Sections 4.1.1 to 4.1.3 live in $1\_data\_processing\_base.ipynb$, for Section 4.1.3 in $2\_data\_processing\_3d.ipynb$ and for 4.1.6 in $3\_data\_split.ipynb$. Additionally, the dataset visualizations in this report can be generated with visualizations.ipynb.
- $model$: The implementation for the deep learning model architecture, as described in Section 4.2, lives here. The model implementation itself can be found in $TennisShotEmbedder.py$ and its model builder parameters can be configured through a configuration file such as $default.yaml$. Finally, this directory includes the Pytorch Dataset implementation for the pre-training and downstream tasks in $data.py$.
- $train$: The code for model pre-training can be found in $PreTrainer.py$ and is executed in $PreTrain.ipynb$. The downstream task training code is similarly included in $DownstreamTask.py$ with its execution in $DownstreamTask.ipynb$.
- $eval$: The evaluation directory contains the 1_eval_pretrained.ipynb notebook, which consists of code that evaluates the prediction quality of the pre-trained model.

Two more main directories are present in the repository. The $data$ directory is where the dataset should be stored and the $models$ directory contains the trained model weights for both the pre-training and downstream tasks.

5. Results

In this section we evaluate the results of our models, both after pre-training and for their performance in the downstream tasks.

5.1. Pre-training Results

In order to evaluate the performance of the pre-trained model, we compute the MSE loss for the predictions on the pre-training test set. The resulting value is 0.0331. As such it is comparable to the final training and validation loss values achieved in Figure 10.

To get a clearer understanding of the predictive capabilities of the model, we perform the following steps. First, we feed the true 3D poses and 2D positions of a test sequence into the model and save the next pose predictions for each frame. After $N_f = 10$ frames, we feed the 3D pose estimate of the last frame into the model along with the true 2D position and, as such, produce predictions for the remaining frames in an iterative fashion. We record the MSE loss for each frame, as visualized in Figure 14.

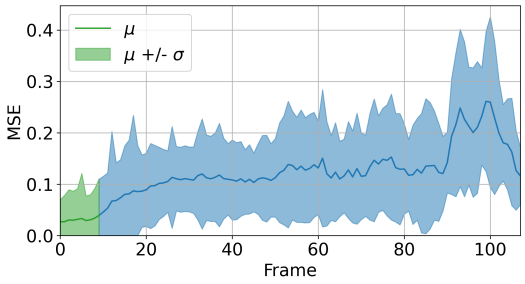
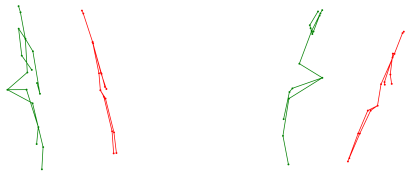


Figure 14: Pre-trained model prediction quality drift.

Initially, the prediction quality worsens linearly with each frame. Around frame 25, corresponding to one second of delay, the error sets into a near constant value up to frame 90 when the prediction quality worsens significantly. Samples for predictions at the first and twenty-fourth frame are given in Figure 15.

We repeat this experiment a number of times with values for N_f of 1, 5, 10 and 25 frames. This ablation study reveals that the quality of the predictions is only for a small amount of iterative frame predictions, with long-term quality drift remaining unaffected.



(a) 1 frame. (b) 24 frames.

Figure 15: Predictions after N frames.
The true pose is in green and the predicted pose in red.

5.2. Downstream Task Results

To evaluate the performance of our model in the downstream task, we consider the results obtained after each phase of the fine-tuning process. In particular, we report the accuracy, precision, recall, and F1 score for each phase, along with their respective confusion matrices.

5.2.1. Pre-Train Phase 1 Results.

After training only the classification network in Phase 1 while freezing the pre-trained layers, the results are as follows:

- Accuracy: 0.6388
- Precision: 0.6386
- Recall: 0.6387
- F1 Score: 0.6353

The confusion matrix for Phase 1 is shown in Figure 16.

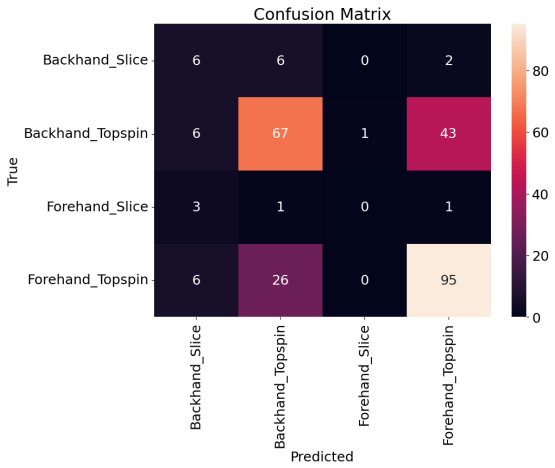


Figure 16: Confusion Matrix for Fine-tuning Phase 1

5.2.2. Pre-Train Phase 2 Results.

After unfreezing all layers and training the entire network in Phase 2, the results improve significantly:

- Accuracy: 0.7605
- Precision: 0.7626
- Recall: 0.7605
- F1 Score: 0.7601

The confusion matrix for Phase 2 is shown in Figure 17.

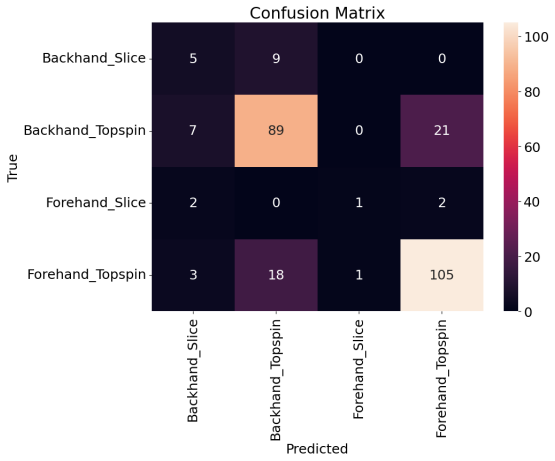


Figure 17: Confusion Matrix for Fine-tuning Phase 2

5.2.3. Analysis of Results.

From the results, it is clear that training the entire network in Phase 2, after training the classification head in isolation (cfr. Phase 1), leads to a boost in all evaluation metrics. Accuracy increases from 0.6388 to 0.7605, and precision, recall, and F1 score also see significant improvements.

The confusion matrices reveal that adjusting the shot representations for the downstream task in Phase 2 helps the model distinguish better between ”Backhand_Topspin” and ”Forehand_Topspin” compared to Phase 1. In particular,

the model's classification of "Backhand_Topspin" improves from 67 correct predictions in Phase 1 to 89 in Phase 2, while that for "Forehand_Topspin" improves from 95 to 105. This highlights the effectiveness of fine-tuning the complete model for downstream tasks.

Overall, even though the results are not perfect, the model performs significantly better than a random baseline. The learned shot representations appear to hold useful information that enables the model to perform well on the downstream hit classification task. While the accuracy could be further improved, these results demonstrate the potential of the pre-trained model in creating meaningful shot embeddings that can be effectively fine-tuned for specific tasks.

6. Conclusion

AI sports coaching is an impactful long term research goal, and learning useful representations of individual sports shots is an key milestone towards that goal. We break down this task into a two step process of workspace reconstruction and motion representation learning from the workspace. We use a mix of classical and deep learning based computer vision to reconstruct the workspace, and graph-based geometric deep learning along with RNN sequence models to learn motion representations. We find that our solution struggles to extrapolate future motion from a given set of frames, but is able to learn representations that can be used for downstream supervised tasks.

7. Future Work

There are two major areas for future work - data collection and model improvements.

With regards to data collection, one can collect a much larger dataset of annotated tennis matches. In addition, one could leverage hand-keypoint extraction models and perform 3D racket and ball tracking to allow for a more detailed graph representation of the workspace. Lastly, multi-modal data that pairs videos and text (captions, coaching feedback, commentary) could enable multi-modal learning in a downstream model.

With regards to model improvements, one should explore using temporal graphs to represent multiple video frames in one graph, and multiplayer graphs to learn relationships between player movements. One should also explore using BERT-style masking and reconstruction of graph nodes. Finally, a multi-modal model could be trained with contrastive learning to connect movement and language.

8. References

[1] Z. Zhao, W. Chai, S. Hao, *et al.*, *A survey of deep learning in sports applications: Perception, comprehension, and decision*, 2023. arXiv: 2307.03353 [cs.CV].

[2] S. Agarwal, T. Duff, M. Lieblich, and R. R. Thomas, "An atlas for the pinhole camera," *Foundations of Computational Mathematics*, vol. 24, no. 1, pp. 227–277, Sep. 2022, ISSN: 1615-3383. DOI: 10.1007/s10208-022-09592-6. [Online]. Available: http://dx.doi.org/10.1007/s10208-022-09592-6.

[3] L. Nogueira, E. C. de Paiva, and G. Silvera, *Towards a unified approach to homography estimation using image features and pixel intensities*, 2022. arXiv: 2202.09716 [cs.CV].

[4] Y. Xiao, Q. Wang, S. Zhang, *et al.*, *Spatialtracker: Tracking any 2d pixels in 3d space*, 2024. arXiv: 2404.04319 [cs.CV].

[5] N. Karaev, I. Rocco, B. Graham, N. Neverova, A. Vedaldi, and C. Rupprecht, *Cotracker: It is better to track together*, 2023. arXiv: 2307.07635 [cs.CV].

[6] W. Zhu, X. Ma, Z. Liu, L. Liu, W. Wu, and Y. Wang, *Motionbert: A unified perspective on learning human motion representations*, 2023. arXiv: 2210.06551 [cs.CV].

[7] M. Dabhi, L. A. Jeni, and S. Lucey, *3d-lfm: Lifting foundation model*, 2024. arXiv: 2312.11894 [cs.CV].

[8] J. Han, J. Cen, L. Wu, *et al.*, *A survey of geometric graph neural networks: Data structures, models and applications*, 2024. arXiv: 2403.00485 [cs.LG].

[9] R. M. Schmidt, *Recurrent neural networks (rnns): A gentle introduction and overview*, 2019. arXiv: 1912.05911 [cs.LG].

[10] C. B. Vennerød, A. Kjærran, and E. S. Bugge, *Long short-term memory rnn*, 2021. arXiv: 2105.06756 [cs.LG].

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014. arXiv: 1412.3555 [cs.NE].

[12] M. Phuong and M. Hutter, *Formal algorithms for transformers*, 2022. arXiv: 2207.09238 [cs.LG].

[13] H. Faulkner and A. Dick, "Tenniset: A dataset for dense fine-grained event recognition, localisation and description," in *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, IEEE, pp. 1–8.

[14] H. Faulkner and A. Dick, *Temporal event annotator*, https://github.com/HaydenFaulkner/TemporalEventAnnotator, 2020.

[15] K. Sergey, *Tennis analysis using deep learning and machine learning*, https://github.com/yastrebksv/TennisProject, 2023.

[16] A. Kulakov, *Open-source monocular python hawkeye for tennis*, https://github.com/ArtLabss/tennis-tracking, 2022.

[17] Y.-C. Huang, I.-N. Liao, C.-H. Chen, T.-U. İk, and W.-C. Peng, *Tracknet: A deep learning network for tracking high-speed and tiny objects in sports applications*, 2019. arXiv: 1907.03698 [cs.LG].

[18] K. Sergey, *Unofficial pytorch implementation of tracknet*, https://github.com/yastrebksv/TrackNet, 2023.

[19] K. Sergey, *Deep learning network for detecting tennis court*, https://github.com/yastrebksv/TennisCourtDetector, 2023.

[20] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV].

[21] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV].

[22] S. E. Whang, Y. Roh, H. Song, and J.-G. Lee, "Data collection and quality challenges in deep learning: A data-centric ai perspective," *The VLDB Journal*, vol. 32, no. 4, pp. 791–813, Jul. 2023, ISSN: 0949-877X. DOI: 10.1007/s00778-022-00775-9. [Online]. Available: https://doi.org/10.1007/s00778-022-00775-9.

[23] MMPose Contributors, *Openmmlab pose estimation toolbox and benchmark*, https://github.com/open-mmlab/mmpose, 2020.

[24] C. Lyu, W. Zhang, H. Huang, *et al.*, *Rtmdet: An empirical study of designing real-time object detectors*, 2022. arXiv: 2212.07784 [cs.CV].

[25] T. Jiang, P. Lu, L. Zhang, *et al.*, *Rtmpose: Real-time multi-person pose estimation based on mmpose*, 2023. arXiv: 2303.07399 [cs.CV].

[26] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, Jul. 2014.

[27] C. S. Catalin Ionescu Fuxin Li, "Latent structured models for human pose estimation," in *International Conference on Computer Vision*, 2011.

[28] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," en, *IEEE Trans Pattern Anal Mach Intell*, vol. 9, no. 5, pp. 698–700, May 1987.

[29] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL].

[30] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[31] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, *Simple and deep graph convolutional networks*, 2020. arXiv: 2007.02133 [cs.LG].

[32] M. E. Peters, M. Neumann, M. Iyyer, *et al.*, *Deep contextualized word representations*, 2018. arXiv: 1802.05365 [cs.CL].

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[34] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245.

9. Appendix

9.1. Individual Student Contributions

TABLE 1: Contributions

UNI	as7092	gyt2107	fps2116
Last name	Safi	Tamer	Sanders
Contribution fraction	33.3 %	33.3 %	33.3 %
What I did 1	Data loader for Pre-training and Downstream Task	3D player pose extraction	Data processing pipeline from collection up to 2D pose extraction.
What I did 2	Code for Training and Evalualating Pretrained Model	Graph construction from pose	Pre-trained model temporal performance drift analysis.
What I did 3	Code for Training and Evaluating Downstream Task	Model builder, config loading, and (jointly) Model implementation	Video and data visualizations and animation.
What I did 4	Contribute to Sections 4.1.6, 4.4 and 5.2 of the report.	Contribute to abstract and Sections 1, 2, 4.1.5, 4.2.1, 6 and 7 of the report.	Contribute to Sections 3, 4.1, 4.2, 4.3, 4.5 and 5.1 of the report.