MatchPoint: Your Computer Vision Tennis Coach

Evan Cheng, Rishi Dange, Isaac Gorelik
Stanford University
450 Jane Stanford Way, Stanford, CA 94305
evcheng@ rdange@ gorelik@stanford.edu

Abstract

MatchPoint is a system that helps amateur tennis players identify the professional tennis player whose forehand tennis stroke most closely matches their own. This helps amateurs discover meaningful counterparts to model their game off of rather than an arbitrary popular player they may know. For each clip comprising of a single forehand shot, we extract four frames which represent the key parts of a tennis stroke. The data then undergoes processing via pose estimation to isolate the important mechnical aspects of each tennis stroke. Beyond the baesline of k-Nearest Neighbors implemented with L2 distance to match inputs to counterpart players, we explored a variety of approaches including Forest Random Forest, Simple Neural Network, Deep Network, and Convolutional Neural Network architectures. We found that given a new clip of a professional player that is in our training set, we are able to match that player to themself with 66.66 percent accuracy using the Deep Network approach. This result is significant given the inherent basic similarities between tennis strokes and the fact that the singular correct player can be identified out of the nineteen players we have in our MatchPoint system approximately two-thirds of the time.

1. Introduction

1.1. Background, Motivation, and Problem Statement

In a highly technical and physical sport that involves individualized nuances in stroke technique, tennis players benefit heavily from video analysis of their own strokes combined with comparisons and analyses with successful professional counterparts. Therefore, we aim to create a pipeline that takes short video clips of a user's forehand and determines the professional tennis player whose forehand most closely resembles it. In doing so, we will enable a richer improvement experience for tennis players to focus on finding their professional-level technical counterparts rather than more famous, trendy, or flashy players whose strokes might not be the right match for a player to try to imitate. This approach may be extended to any other tennis stroke (serve, backhand, volley, slice, etc.), given appropriate data from professional players. Furthermore, this pipeline may be applicable to many other sports and movement-specific applications as well.

The input to the MatchPoint pipeline is an ordered set of four images taken at the four key frames of a forehand stroke, which should be extracted from a video of a forehand swing taken from the front, facing the player. For optimal results, the images should match the specifications in Figure 1:

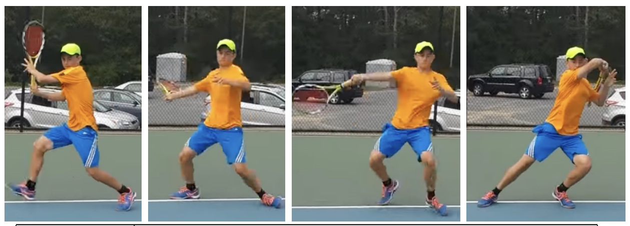


Image 1	Highest wrist position during backswing
Image 2	Lowest wrist position during wrist drop
Image 3	Contact point
Image 4	Conclusion of follow through

Figure 1. Specifications of input images for the MatchPoint pipeline, with an example of four key frames from a clip of Isaac's forehand in 2015

Given four input images of this form, MatchPoint will identify the professional player (out of the 19 candidate players in our dataset) whose forehand most closely resembles the input clip. MatchPoint performs this search first over the per-frame level, identifying the individual professional frames that most closely matches each of the four frames provided in the input, and then aggregates these results to provide a final suggestion on which player to mimic.

1.2. Further Tennis Context

A player's preferred forehand hitting style can be broken down into many components, such as their grip on the racket, how wide their base is, how deep the knee bend is, hip rotation at preparation and contact, non-dominant arm movement, and most notably, the depth of their elbow bend at contact. Whether a player hits their forehand with a bent elbow (called a “double-bend” forehand – notable example being current world no. 1 Novak Djokovic) or a straight elbow (called a “straight-arm” forehand – notable example being former world no. 1 Roger Federer) is a key point of discussion for tennis players and coaches alike. Interestingly, many amateur players are oblivious to their preferred elbow bend style, and will simply arbitrarily copy one of the game’s many greats, which may inadvertently lead their tennis improvement astray. Our hope is that MatchPoint might help mitigate this issue, and indeed, our results on amateur video input seems promising. For example, Isaac, shown in Figure 1, is matched to Holger Rune. Both players employ a straight-arm forehand and a deep knee bend.

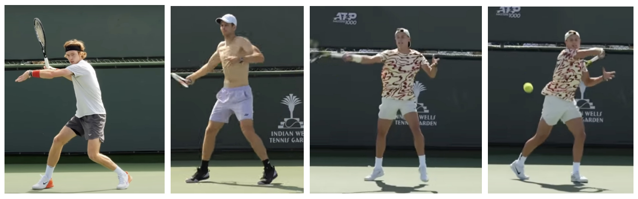


Figure 2. MatchPoint professional counterpart frames for the amateur player, Isaac, shown in Figure 1. Left to right: Andrey Rublev, Hubert Hurkacz, Holger Rune. With two strongly weighted votes, the model’s coaching suggestion would be to mimic Holger Rune. As an aside, note that each input frame is correctly matched to a frame from the corresponding swing phase.

2. Related Works

There exists limited past work that performs stroke comparison between tennis players. Both Bloom et al. [8] in their 2003 paper and Bacic et al. [5] in their 2022 paper perform classification by four broad categories (forehand, backhand, serve, no stroke), while a 2020 paper by Baily et al. [6] presents a method to compare the joint movements of different professional tennis players during their serves. Each uses different methods to distill the essence of a tennis stroke from visual data: Bloom et al. use a bounding box and low-pass filtering to isolate the overall shape of the player on the court, and Bacic et al. and Baily et al. use more modern pose estimation systems of OpenPose and Alpha Pose, respectively. The performance of the systems in the two more recent far exceeds the performance of the Bloom et al. method in large part because use of pose estimation to isolate individual joints on the body is more resistant to biases like player clothing or size that do not reflect the action on the court.

A 2022 comparative analysis of pose estimation models by Jo et al [11]. shows that MoveNet Lightning [10] is the fastest pose estimation model while OpenPose is the most accurate one, yet by far the slowest in part due to advanced features like recognizing multiple people within a single frame. Because these advanced features are not needed for our dataset and because the accuracy between OpenPose and MoveNet Thunder is comparable, we chose to use MoveNet Thunder which combines most of the accuracy of OpenPose with most of the efficiency of MoveNet Lightning. (All references to MoveNet from hereforth refer to MoveNet Thunder.) Another 2022 analysis by Washabaugh et al. [13] showed that MoveNet Thunder was most accurate for measuring hip kinematics, which is especially important in the tennis setting because hip rotation is a major differentiator in the strokes of different players.

In terms of classification of players based on image data, there exist a wide range of methods used, from Baily et al., which simply relies on Euclidean distance between key points to perform classification, to Bacic who in a 2016 paper [7] proposes use of an Echo State Network (ESN) for action shot classification whereby a large, randomly initialized reservoir remains unchanged. There also exist more canonical neural network approaches like Ganser et al. [9] and convolutional neural network (CNN) approaches like those used in Anand et al. [4] and Skublewska-Paszkowska et al. [12], which show promising results on analysis of tennis action, albeit with much broader and easier-to-achieve classification goals than ours: to designate tennis shots as one of five shot types and to distinguish strokes from similar racket sports. We adopt the use of Euclidean distance and k-Nearest Neighbors (KNN) from Baily et al. as our baseline approach. While the results from Anand et al. and Skublewska-Paszkowska et al. indicate promise for the CNN approach, our synthesizing of image data into joint estimation key points makes the notion of spatial locality as applied in CNNs more dubious. It is therefore possible that a deep neural network approach without convolutional layers would out-perform a CNN in our setting. Therefore, we construct both simple neural network and CNN architectures to find the ideal architecture for MatchPoint.

3. Dataset

Among our contributions for this paper is our work collecting data for the forehand dataset, which consists of a total of 452 forehand frames for 19 professional men’s tennis players. This data was collected by hand from three large front-facing forehand compilations on YouTube [1] [2] [3]. To collect the images, we selected a continuous forehand clip in the video and captured the four key photo frames at the essential moments of the stroke, as shown in Figure 3.

For training data, we used 5 distinct forehand clips per player, for each of the 19 players, resulting in a total of 95 clips. Each clip is used to generate the four key forehand frames, yielding a total of 380 images in the training set.

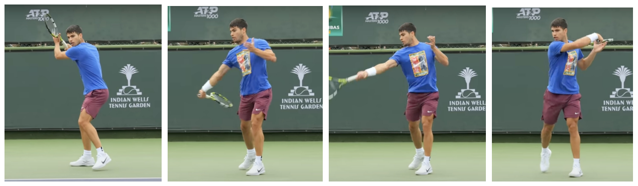


Figure 3. Four key frames of a Carlos Alcaraz forehand swing. Left to right: 1) highest point of backswing, 2) lowest point of wrist drop, 3) contact point, 4) conclusion of follow through.

When training our neural model, we used a validation set, split during runtime, by randomly selecting one of the 5 continuous "clips" (i.e. ordered set of four images) for each player, and using the remaining four clips per player to train the model. Our heldout test set consisted of the remaining 18 clips unused by the training set (72 images). Lastly, for amateur analysis, we used one clip each of our own forehand strokes and an additional old video of Isaac (shown in Figure 1), and captured the four key frames per shot.

4. Methods

4.1. Keypoint Extraction Using MoveNet

For the purposes of a technical analysis of one's forehand, an image frame may potentially carry excessive and unnecessary information, such as the color of a player's clothing, the color of the tennis court and their immediate surroundings, the player's facial structure, etc. These unnecessary pixels are at best simply a waste of computational resources and image processing power, but at worst, these pixels may become misleading noise which harms our model's performance on the task of technical tennis comparisons. Therefore, in order to avoid learning the superficial features unrelated to the technical analysis of a player's forehand, we preprocess the stroke image frames to extract the positions of the 17 biomechanical keypoints (positions of knees, hips, shoulders, etc) present in the image using the MoveNet pose detection model [10]. This preprocessing is done before performing any algorithmic comparisons or learning techniques. By using keypoints rather than image data, we extract the most essential information needed about a player's swing during that frame, while eliminating much of the irrelevant information that may cause incorrect pattern matching. Furthermore, this drastically reduces the dimension of our data, where images, initially represented as tensors with thousands of real-valued points, are condensed to a single 3x17 matrix, containg the x and y coordinates of each keypoint and a "confidence score" assigned to the keypoint by the MoveNet pose detection algorithm.

As part of preprocessing, we also flip the keypoint coordinates along the x-axis for left-handed players to ensure consistency with right-handed players. In the absence of this step, MatchPoint would only produce matches between players of the same handedness, since processing is performed on the location of the extracted key points and the distance between key points for a right-handed stroke and a left-handed stroke is too large to produce matches otherwise.

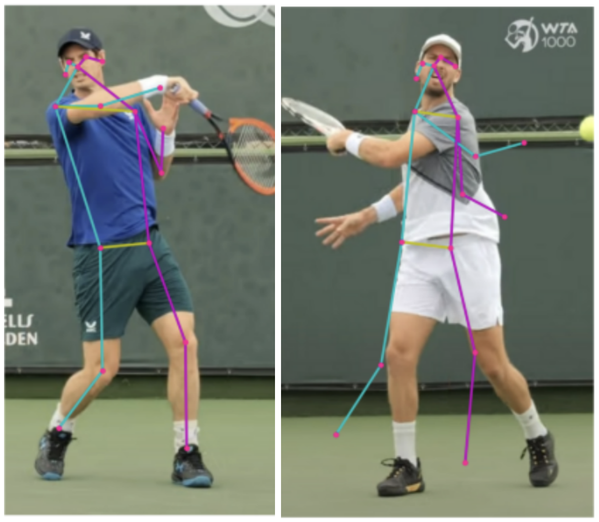


Figure 4. MoveNet keypoints overlaid on images of Andy Murray (left) and Cam Norrie (right) at the follow-through phase of their forehand swings. Norrie's left-handed keypoints are reflected to make for meaningful comparisons with right-handed players.

4.2. Keypoint Normalization

Proper normalization of the obtained MoveNet keypoint positions proved to be essential for our model's performance, as the keypoint positions given by the MoveNet algorithm are relative to the height and width of the provided image. Given that the ultimate goal of MatchPoint is to provide guidance to tennis players providing their own images, it was essential that our approach would be robust to changes in image size, camera orientation, and relative position of the player in the shot. Furthermore, our own obtained images in the training data contained images of various sizes.

Our first normalization approach was to use minmax normalization across both x and y axes, scaling the positions of the points so that they fit in and span the unit box. This proved to be an improvement over unnormalized keypoint positions, as the minmax normalization ensured that the keypoints spanned the unit box, regardless of the size of the original image or the player's position in that image.

Our second normalization approach, which we called torso normalization, was to shift the keypoints so that they are centerered about the player's left hip, and scale the keypoint distances so that the length of a player's torso (distance between left hip – now the origin – to the left shoulder) equals one unit. The rationale behind this approach was the observation that for the non-follow-through frames, the right hand was typically the left-most point in the image. Therefore, with minmax normalization, this point would almost always have an x coordinate of zero. We suspected that this point carried essential information about a player's technique, so we suspected that artificially restricting this point to have an x coordinate of zero would harm our model's generalizability.

Ultimately, both normalization approaches led to improved performance in our baseline nearest-neighbors comparison task over the unnormalized points, and considerably improved the capacity of nearest-neighbors to yield meaningful results for our unseen amateur clips. Given its superior performance and its intential design to be better suited for forehand analysis, we used torso normalization for all downstream tasks.

4.3. Dataset Visualization

Having extracted and normalized the keypoints, we used principal components analysis (PCA) and T-distributed Stochastic Neighbor Embeddings (t-SNE) to reduce our data to two dimensions and investigate the underlying structure of the space of our professional forehand keypoints. The hope was that two levels of clusters would emerge: 1) large, global clusters separating the four key frame points, and 2) smaller, more granular clusters of distinct play styles. The unnormalized keypoints are visualized in Figure 5

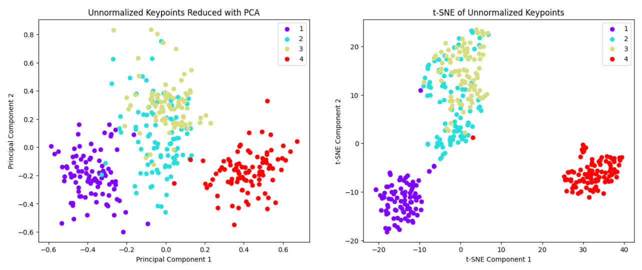


Figure 5. PCA (left) and t-SNE (right) visualizations of unnormalized keypoints. Frames 1 and 4 form distinct clusters, but 2 and 3 are not easily distinguishable.

In both techniques, frames 1 and 4, at opposite ends of the swing, were easily separated, but there was considerable overlap between frames 2 and 3. This overlap was still present, though slightly mitigated, through torso normalization, as shown in Figure 6

4.4. Baseline Method

Our baseline method is using the k-Nearest Neighbors Classifier (KNN) on the normalized data to identify which player in our dataset most closely matches the user based on the four input images provided by the user. Since MoveNet produces key points for each image as 17x2 coordinate arrays, we use L2 distance to determine the distance between

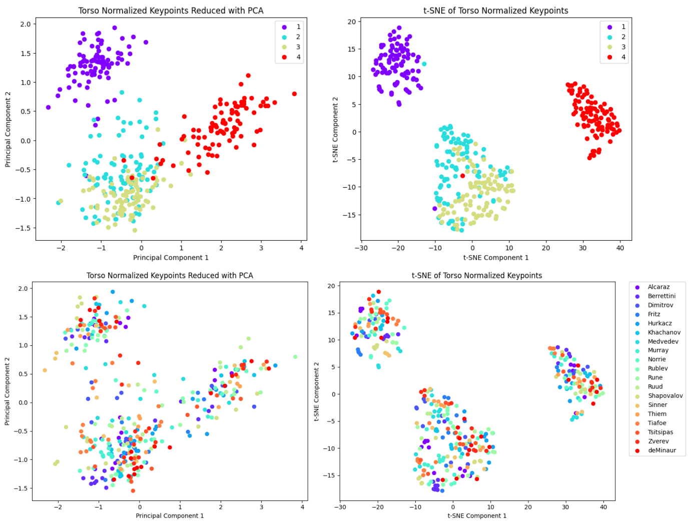


Figure 6. PCA (left) and t-SNE (right) visualizations of torso-normalized keypoints used in the MatchPoint approach, colored by frame type (top) and player (bottom). Frames 1 and 4 remain distinctly separated, and frames 2 and 3 seem slightly more distinct.

frames as a simple and natural distance metric. The KNN method identifies the k closest training frames to the input frame based on L2 distance, then has them vote on the player and frame that most matches the input image is. For example, for k=3 the KNN method may identify Alcaraz-A3 (Alcaraz's contact point in training sample A of Alcaraz), Medvedev-B3 (Medvedev's contact point in training sample B of Medvedev), and Alcaraz-B3 (Alcaraz's contact point in training sample B of Alcaraz) as the three closest neighbors; Alcaraz-3 (Alcaraz's contact point) would then be identified as the most similar player-frame combination to the input image. To return a single most similar player for a set of four input images, the baseline method then chooses the player that appears most often as a k-closest neighbor across the four images. Using KNN, we sweep over different k values to determine the best possible performance of our baseline method, as measured by the percentage of test input images for which the KNN algorithm correctly identifies the player in question.

4.5. Player Classification: Advanced Strategy

As a much more advanced means of performing player classification, we utilize more complex machine learning techniques. We employ the same preprocessing techniques as with our baseline model (namely, keypoint extraction using MoveNet and keypoint normalization) to begin with a size-normalized 17x2 coordinate array for each frame of a particular player's stroke. For every player in our training/validation dataset, we have several groupings of four images that correspond to the pre-defined and aforementioned four critical locations in the player's stroke.

In order to ensure that every critical location has a vote in determining the professional player that is the best match, we build four separate classifiers (all of the same type, and not to be confused with the four different classifier types that we implement below to test which classifier type works the best). Each classifier corresponds to one of the critical locations in the player's stroke and, when performing a prediction, outputs a probability distribution representing the likelihood of the critical location image corresponding to each player in the training dataset. Each classifier is only trained on images from the training dataset that correspond to the designated critical location in the player's stroke. When performing prediction on the validation and testing datasets, we use each classifier to create a probability vector for its corresponding input critical location image, and then we aggregate these probabilities using a simple average. We use a mean aggregator here to ensure that our final aggregated prediction vector is itself a probability distribution over the professional players hosted within the training dataset. Figure 4.5 illustrates the functionality behind our model framework.

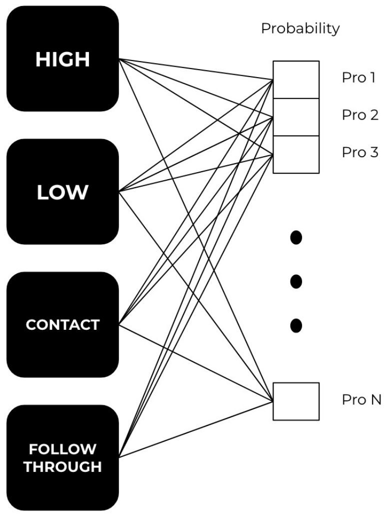


Figure 7. Aggregation of classifier output to create a final prediction probability distribution for validation and test datasets.

We implement the following four different classifier types and evaluate our findings from applying each of them through experimentation.

4.5.1 Classifier 1: Random Forest

Here we use a `RandomForestClassifier` from `sklearn.ensemble`. It uses 100 decision tree classifiers followed by averaging to learn different aspects of the classification task. We utilize this classifier because of its ability to learn different aspects and different patterns of the classification task at the same time. We use the `predict_proba` function in order to obtain not only the predicted professional player for the critical location image but also the probabilities the classifier has attributed to each of the possible professional player outputs (as required by our overall structure).

4.5.2 Classifier 2: Simple Neural Network

As our second type of classifier with which we want to experiment, we implement a rather simple neural network. It consists of two fully-connected layers with a ReLU (rectified linear unit) nonlinearity in between, defined by

$$\text{ReLU}(x) = \max(0, x).$$

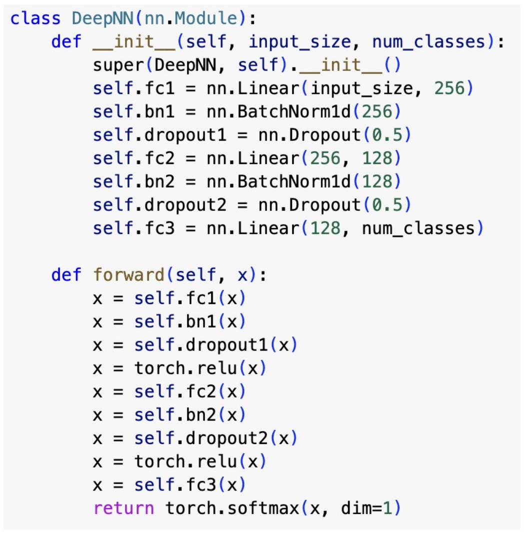
. To arrive at a probability distribution, we use the Softmax function, defined by

$$\text{Softmax}(x_i) = \frac{\exp{(x_i)}}{\sum_j \exp{(x_j)}}.$$

When training the neural network, we use the `Adam` optimizer and a basic cross entropy loss as our classification loss. We iterate over 600 epochs and use a learning rate of 0.01, both being hyperparameters that we have selected empirically based on our training and validation results.

4.5.3 Classifier 3: Deep Neural Network

As our third type of classifier to explore, we implement a deeper, more complex neural network than the previously described one.



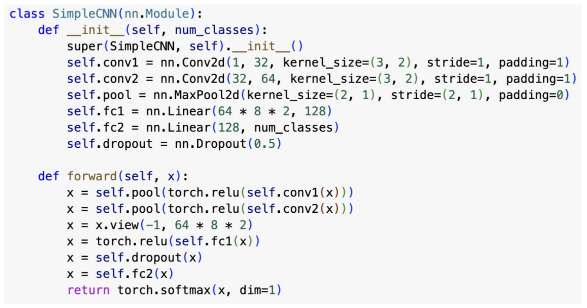
```python
class DeepNN(nn.Module):
    def __init__(self, input_size, num_classes):
        super(DeepNN, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.bn1 = nn.BatchNorm1d(256)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256, 128)
        self.bn2 = nn.BatchNorm1d(128)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.dropout1(x)
        x = torch.relu(x)
        x = self.fc2(x)
        x = self.bn2(x)
        x = self.dropout2(x)
        x = torch.relu(x)
        x = self.fc3(x)
        return torch.softmax(x, dim=1)
```

Figure 8. Architecture of the deep neural network used as a classifier.

Figure 4.5.3 displays the architecture of this deep neural network classifier. The hidden layers enable it to learn more complex patterns in the data that a simple fully-connected layer cannot comprehend. We use a Scikit-Learn `StandardScaler` to normalize the input data, and as before, we use the `Adam` optimizer along with cross-entropy loss for our training process. As a noteworthy difference to our simpler neural network, we implement learning rate scheduling in the form of a 0.8 factor decrease every 20 epochs, and we run for 500 epochs but stop early if we do not see improvement for 50 consecutive epochs (and save the best model accordingly).

4.5.4 Classifier 4: Convolutional Neural Network

As a means of attempting to benefit from the spatial nature of our 17x2 coordinate data, we implement as our fourth classifier a convolutional neural network.



```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=(3, 2), stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=(3, 2), stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=(2, 1), stride=(2, 1), padding=0)
        self.fc1 = nn.Linear(64 * 8 * 2, 128)
        self.fc2 = nn.Linear(128, num_classes)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 2)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return torch.softmax(x, dim=1)
```

Figure 9. Architecture of the convolutional neural network used as a classifier.

Figure 4.5.4 displays the architecture of this convolutional neural network classifier. As before, we use the `Adam` optimizer along with cross-entropy loss for our training process. We implement learning rate scheduling in the form of a 0.8 factor decrease every 25 epochs, and we run for all 500 epochs without early stopping (but we save the best model along the way). Considering that we do not have thousands of datapoints at this point in time (and that our 17x2 coordinates are not a large number of evenly-spaced pixels), we do not necessarily anticipate great performance from this architecture. Nevertheless, we implement it and experiment on it.

5. Experiments

5.1. Evaluation

For evaluation, we will first confirm that additional images of a player from the training set hitting the same shot are matched to that player. This self-classification accuracy will be a key quantitative component in the analysis of our model's effectiveness and is feasible since the same players show up in our training and test sets. As explained previously, the validation set and training set differ in that the validation set comes from the same practice session for the player as the training data (noting critically that we did separate out the validation images to ensure their integrity), while the training set comes from videos taken a year later (so would perhaps be more akin to unseen data). We will also qualitatively evaluate the mismatches arising from our experiments to determine the cause of error and from which step of the MatchPoint process the problems may arise.

5.2. Baseline Results

Sweeping across different k values to run the baseline KNN algorithm on the test set, we observed the following baseline results. At k=5, at which point we see the best results, 56 percent of test inputs result in the identification of the correct corresponding player using KNN. We look to improve upon these baseline results in our advanced methods.

5.3. Advanced Results

After completing the training, validation, and testing procedures across all of our datasets, we generated accuracy metrics to inform us of how well they performed. Specifically, for each type of classifier (random forest, simple neural network, deep neural network, and convolutional neural network), we compute the fraction of instances (separately for the validation set and the test set) in which our aggregated 4-location scheme produced the correct name for the professional player. We call this accuracy our Top1 Accuracy, as it determines whether or not the highest probability prediction matches the ground truth. To further assess the efficacy of our models, we use the Top3 Accuracy metric, in which we see for what percent of validation (or test) data the ground truth is contained within the set of the top 3 highest probability predictions. Table 1 summarizes the results we obtain.

In all cases, we vastly exceed the accuracy of 5.3 percent that we would expect if we were to randomly select a player from the training set as the best match.

5.3.1 Validation Set Performance

On the validation set, we see that the simple neural network achieved the highest Top1 Accuracy at 94.74%, followed very closely by the deep neural network (89.74%). We also see that these two neural network models exhibited tremendous Top3 Accuracy performance, with the simple neural network achieving 100% perfection. The random forest classifier did considerably worse in terms of Top1 Accuracy but showed sizeable improvement when it came to Top3 Accuracy, matching the Top3 Accuracy of the deep neural network. The convolutional neural network approach was very clearly outperformed by the other three classifier types.

	Validation Set		Test Set	
Model	**Top1 Accuracy**	**Top3 Accuracy**	**Top1 Accuracy**	**Top3 Accuracy**
Random Forest	68.42%	94.74%	61.11%	66.67%
Simple NN	94.74%	100%	61.11%	61.11%
Deep NN	89.47%	94.74%	66.67%	72.22%
CNN	52.63%	78.95%	38.89%	55.56%

Table 1. Comparison of Top1 and Top3 Accuracy for different models on validation and test sets

5.3.2 Test Set Performance

On the test set, we see that the deep neural network architecture was the clear frontrunner, with a Top1 Accuracy of 66.67% and a Top3 accuracy of 72.22%. The simple neural network and random forest approaches both performed decently well, with only the random forest approach showing improvement from Top1 to Top3 Accuracy. As with the validation set, we see that the aforementioned three classifier types outperformed the convolutional neural network architecture.

5.3.3 Discussion and Qualitative Findings

In our results we see very strong performance from both neural network candidates. The simple neural network was outstanding with respect to the validation set, showing its ability to generalize to new but similar images to what it has seen in training. The deep neural network appears to perform similarly (slightly worse) on validation data, but clearly is more robust in that it shows considerable improvement over the simple neural network model when it comes to the testing data. Going forward, our deep neural network approach would be the one we would use for demonstration purposes as it clearly generalizes the best to unseen data.

The random forest approach seems to benefit the most from the incorporation of the Top3 Accuracy metric. This result is perhaps an indication of its ability to cluster players decently well and determine which players might be close in style to a particular player without pinpointing a single player. As expected, the convolutional architecture underperformed, likely due to the need for more data and better hyperparameters.

Qualitatively, we notice that running shots cause the most errors (see Figure 10). We noticed this when we manually went through to view the cause of the mismatches we experienced, especially those that arose from our most successful (i.e. the neural network) approaches. This indicates that refined data pre-processing may yield more benefits to the accuracy of MatchPoint than improvements to the classification model itself; in almost any case, a player whose data mostly consists of stationary shots is matched to themself when the corresopnding test data corresponding is also a stationary shot. This is similar to the potential issue we would have encountered had we not flipped the images for left-handed players.

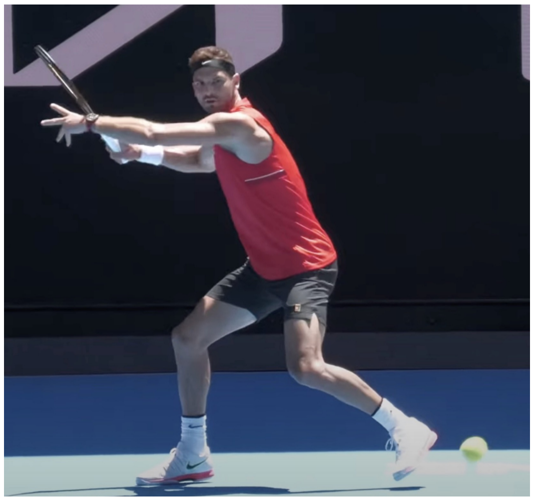


Figure 10. Example of a running forehand shot by Grigor Dimitrov that was misclassified as Alex Zverev.

6. Conclusion

MatchPoint allows users to input a very limited amount amount of images of their tennis forehand shot as input and matches those users with high accuracy to the professional player whose tennis stroke is most similar to theirs. After evaluating many different approaches including Random Forest, Simple NN, Deep NN, and CNN, we found that the Deep NN performs best on the test data, with 66.67 percent accuracy in identifying the one player most similar to the test input. This is a significant result, since many tennis strokes share major similarities and with a two-thirds chance we are able to identify the one professional out of 19 candidate players whose forehand most closely matches the input clip. The correct player is furthermore found in the top three closest matches 72.22 percent of the time. These numbers represent a meaningful improvement over the 56 percent accuracy of the baseline KNN results.

Further work would involve firstly expanding the training and data sets to further validate our results. We may also look into exploring further granularity in categorizing shots up-front (e.g. running shots versus stable shots), as this noise in the position of players during different types of forehand shots we believe has a large impact on how they are classifiers. We also hope to explore using video recognition to automatically capture the four key frames needed to comprise a data point for MoveNet, which would in turn help with the first point of acquiring more data as the data acquisition process would no longer need to be done by hand. Nonetheless, even with the current system and its results, MatchPoint stands to benefit amateur players by helping them model their play off players with similar mechanics to themselves rather than simply their favorite player whose mechanics may not align with theirs.

References

[1] Forehand compilation — slow motion 2023. `https://youtu.be/_7xV_CE8y28?si=GwZF9G9uHVTih80o`, 2023.

[2] Forehand compilation — slow motion 2023 (part 2). `https://youtu.be/PoF1iCUIGtc?si=esUV82E4KnSLBtfj`, 2023.

[3] Forehand compilation — slow motion 2024. `https://youtu.be/928wJjWeVyk?si=XHUA92j1hY9qIdZR`, 2024.

[4] A. Anand, M. Sharma, R. Srivastava, L. Kaligounder, and D. Prakash. Wearable motion sensor based analysis of swing sports. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 261–267, 2017.

[5] B. Baić and I. Bandara. Tennis strokes recognition from generated stick figure video overlays. In *VISIGRAPP*, 2022.

[6] L. Bailey, N. Truong, and P. Nguyen. Stroke comparison between professional tennis players and amateur players using advanced computer vision. 2020.

[7] B. Bačić. Echo state network ensemble for human motion data temporal phasing: A case study on tennis forehands. In *Neural Information Processing. ICONIP 2016*, volume 9950 of *Lecture Notes in Computer Science*. Springer, 2016.

[8] T. Bloom and A. Bradley. Player tracking and stroke recognition in tennis video. 1, 03 2003.

[9] A. Ganser, B. Hollaus, and S. Stabinger. Classification of tennis shots with a neural network approach. *Sensors*, 21:5703, 08 2021.

[10] T. Hub. Movenet: Ultra fast and accurate pose detection model. `https://www.tensorflow.org/hub/tutorials/movenet`, 2024. Accessed: 2024-05-17.

[11] B. Jo and S. Kim. Comparative analysis of openpose, posenet, and movenet models for pose estimation in mobile devices. *Traitement du Signal*, 39(1):119–124, 2022.

[12] M. Skublewska-Paszkowska, E. Lukasik, B. Szydlowski, J. Smolka, and P. Powroznik. Recognition of tennis shots using convolutional neural networks based on three-dimensional data. In A. Gruca, T. Czachórski, S. Deorowicz, K. Harežlak, and A. Piotrowska, editors, *Man-Machine Interactions 6. ICMMI 2019*, volume 1061 of *Advances in Intelligent Systems and Computing*, Cham, 2020. Springer.

[13] E. Washabaugh, T. Shanmugam, R. Ranganathan, and C. Krishnan. Comparing the accuracy of open-source pose estimation methods for measuring gait kinematics. In *Gait Posture*, volume 97, pages 188–195, 2022.

Appendix

Professional players represented in the MatchPoint dataset, sorted alphabetically. Note that left-handed player's marked with (L) have their forehand images reflected so that technical nuances can be captured, rather than the player's dominant handedness:

Carlos Alcaraz, Matteo Berrettini, Grigor Dimitrov, Taylor Fritz, Hubert Hurkacz, Karen Khachanov, Daniil Medvedev, Andy Murray, Cam Norrie (L), Andrey Rublev, Holger Rune, Casper Ruud, Dennis Shapovalov (L), Jannik Sinner, Dominic Thiem, Frances Tiafoe, Stefanos Tsitsipas, Sasha Zverev, Alex de Minaur.

7. Contributions and Acknowledgements

All three group members contributed equally to the project and report. Evan had a slight focus on data collection and baseline results, Rishi had a slight focus on exploring neural network models, and Isaac had a slight focus on data processing and random forest, but all members had a hand in all aspects. We would like to thank the CS 231N course staff, especially Abhijit, for their support with our project.