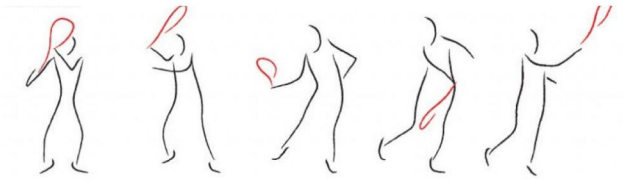
CS 230

Deep Learning for Tennis Stroke Classification Using Tech Wearables

Authors: Tristan Gosakti (tgosakti@stanford.edu) and Ellen Roper (eroper@stanford.edu)

Abstract:

Motion capture technology is becoming increasingly prevalent, offering a myriad of applications. As a result, TuringSense, a Santa Clara-based tech wearable company, created a product which uses motion capture to train athletes. Standard motion capture is often optical in nature, but TuringSense replaces this approach with digital technology, opting to use their own motion sensors instead. Currently, TuringSense uses mathematical models from the motion sensors' data to classify tennis strokes. This report details our implementation of a single layer neural network, a deep neural network, and a convolutional neural network to classify 5 different tennis stroke types with final test set accuracies of 0.96, 0.97 and 0.98 respectively.



Introduction

Currently, TuringSense uses their own mathematical models (derived from trial and error) to process raw data from 6 sensors attached to different parts of the human body and the tennis racket for tennis stroke classification. They are looking to improve the accuracy of this model, so we have offered to use deep learning algorithms in an attempt to do this. We cannot disclose the metrics of their algorithms and the final purposes of these sensors due to an NDA, but it is clear that providing data feedback to their users is one of TuringSense's most important values, so this project's result is very significant.

The input to the algorithm is processed data from 6 of TuringSense' sensors projected over a time span. We used linear regression (single layer NN), a deep-layer (four layer NN), and a convolutional neural network to produce outputs that classify the stroke into one of the following five strokes: (1) forehand flat, (2) forehand topspin, (3) backhand slice, (4) backhand topspin, and (5) forehand slice. Because of TuringSense's limited time and resources, we were only able to use training data from a single person, as we collected data ourselves.

Related Work

Most work regarding tennis stroke classification enlists the use of video capturing technology. In this case, there are some classifiers that make use of non deep-learning algorithms such as the SVM (Support Vector Machine) classifier, "used to identify the strokes based on the extracted features" of the "player's skeleton" (Shah, Hitesh, et al). While this paper details a SVM classifier that has an accuracy of around 97%, limitations of this method include the high costs and computing power necessary to continually track the player through a video and extract their features. Furthermore, this algorithm only classified between two easily distinguishable strokes, the forehand and backhand, while our algorithm aims to classify intricate subtleties (e.g. differentiating between a forehand slice and a forehand with topspin). In this case they used raw sensor data instead of processed angled data as our model does.

In other works, (Connaghan, Damien, et al.), an approach using body sensors are used. The sensors used in this paper captures accelerometer, gyroscope and magnetometer data, similar to TuringSense' sensors. This work uses SVM classifiers and K-means nearest neighbor clustering algorithms to classify their data. Since SVM classifiers are non-probabilistic binary linear classifiers, this method's complexity mirrors that of our initial single layer baseline model.

This paper managed to reach an accuracy of 90% with a "single inertial measuring unit attached to a player's forearm." However, like many other papers they only classified between two easily discernible strokes - forehand and backhand.

One of the more promising papers used CNNs for wearable tech, which highlighted its "novelty" in using this technique. Authors of this paper claim that their approach "outperforms state-of-the-arts in terms of recognition accuracy and computational cost." (Jiang, Wenchao, and Zhaozheng Yin) Their approach consists of transferring "all time-series signals from the accelerometer and gyroscope into a new activity image", from which they use a CNN on, similar to what we did. However, this paper only tested 3 possible outcomes – sitting, walking and standing.

Dataset and Features

TuringSense does not currently have a database of data from different strokes, so data collecting was a non-trivial part of this project. We used TuringSense's six sensors, one on the tennis racket and five on the arms and chest. Each sensor consists of accelerometers, gyrometers, and quaternions and records data at 100Hz (100 updates a second, so each time tick is 10ms). The raw sensor data is converted to three 'joint angle' data points using TuringSense's existing algorithms, which results in 18 data points per 10ms. Alongside this data in the form of large text files, TuringSense's technology provides us with metadata detailing the beginning and end timestamps of each stroke so that we can isolate sensor data that reflects each stroke.

Data collection sessions were divided into batches – 10 shots of the same stroke were performed in each batch (to avoid a technical issue with their sensors). Because of TuringSense's limited time and resources, they allocated a single person whose strokes we trained and tested our data on. Data collection happened over the span of two days, and we collected 10 batches of 10 samples each (10 samples * 10 batches = 100) for 5 different stroke types. On some occasion, 11 sample shots were accidentally recorded into a batch – we ended up with 507 tennis strokes.

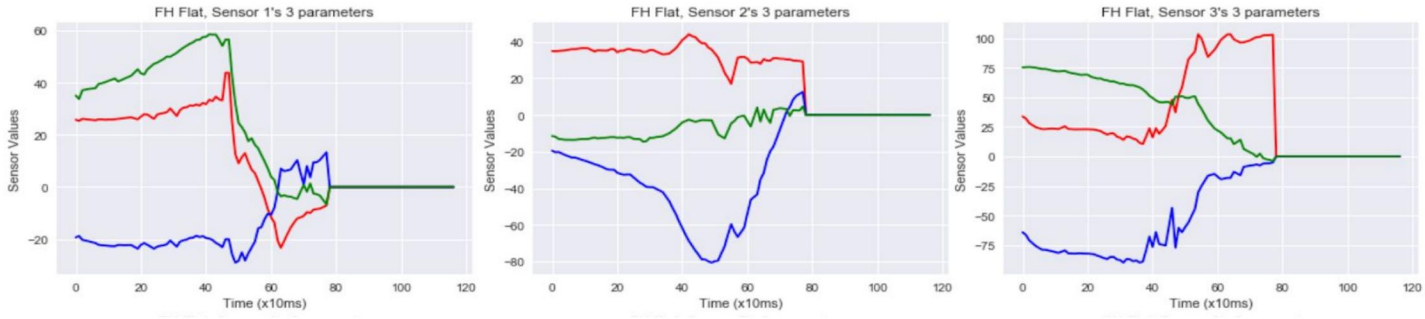


Figure 1. Joint angle data from three (out of six) sensors for a forehand flat stroke over time.

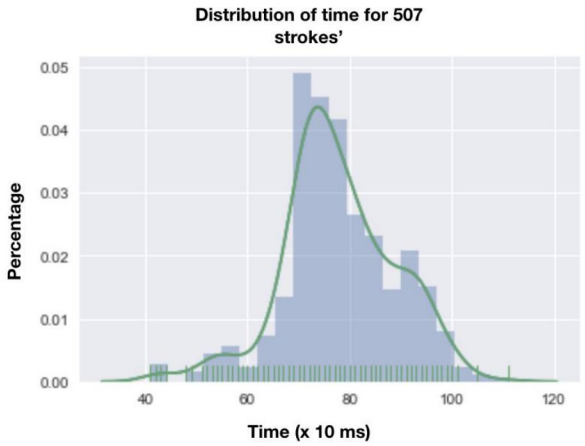


Figure 2. After performing analysis on 507 strokes, we found that the longest stroke length is 111 ticks (1.11 seconds). The distribution has mean of 77.78 ticks (777.8 ms) and standard deviation of 11.00 ticks (110 ms).

Because we had to feed in a predictable input size, we padded the data from each stroke. For the single layer and four layer neural net, data points were stacked vertically and each stroke was padded with zeros until all input vectors were length 2000. This allowed us to accommodate the longest stroke (111 ticks * 18 data/tick = 1998 data points). For

our convolutional model, sensor data within each individual timestamp was stacked vertically and appended horizontally into an 18 data point x 111 timestamps data "image." We shuffled the data and separated it into 407 samples for the train set, 50 samples for the dev set and 50 samples for the test set. Because of the limited number of training samples, we then performed some data augmentation on the training set. Detailed analysis showed that different sensors and different strokes had predictable trends but different individual distributions. The most plausible method of data augmentation was to incorporate noise into the sensor data, which we accomplished by adding or subtracting element-wise noise from each stroke data in a vector of random normally distributed values. We experimented with different standard deviations for this distribution, all centered around 0, to ensure that our model was adaptable to different types and magnitudes of noise.

As seen below, the different sensors show different distributions of data. With further analysis of other strokes, we also tried adding the same bias to every data point as augmentation. This is because, as clearly shown by the graph on the left, the strokes appeared to be cookie-cutter versions of themselves translated up and down the y axis by a certain amount. As TuringSense's sensors seem very accurate, this appeared to be a valid approach to augmenting data. Through inspection of many other sensors and strokes, the broadcasted bias was drawn from a normal distribution of mean 0 and variance of 1, and the CNN ultimately performed consistently with augmented data of this type as well as with element-wise noise.

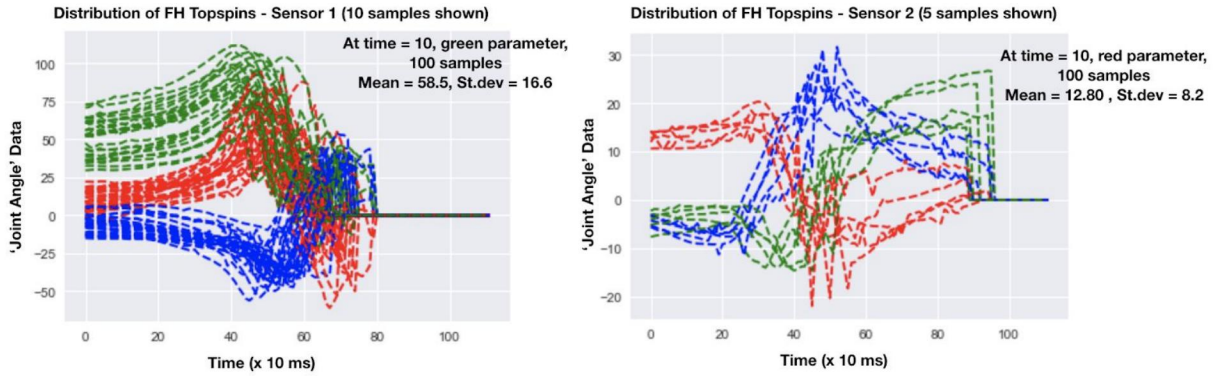


Figure 3. Data for the forehand topspin stroke from sensors 1 and 2, showing a predictable trend over several strokes.

Methods

The initial model we tested was a basic neural network with one hidden layer. We used a softmax activation function with cross entropy loss function. Through trial and error, we settled on 500 neurons in the hidden layer with a learning rate of 0.1. This turned out to be adequate enough to produce accuracy of 96% in the test set. Further tuning did not result in anything higher. The input data for this model was a flattened vector of data values of size (2000, 1). It went through the 500-unit layer, and ultimately through the softmax function, which resulted in an output vector containing the classification probabilities, which was compared with an initial one-hot label vector.

The second model we tried was a multi-layer deep network, with the same input of a (2000,1) vector and a cross entropy loss function with a softmax activation function. We settled on four layers because the addition of more after the initial four stopped improving accuracy regardless of tuning, and because the accuracy of 97% was working well already. Between each layer, we used a ReLU activation function, as the input is decidedly nonlinear. Each sensor outputs data which changes over time at different rates, meaning that there are noticeable non-linearities in the data. ReLU activation accounts for different changes in the slope of the data at different points in time, giving a more accurate threshold for the neurons to activate.

In a further attempt to increase the accuracy of our results, we used a convolutional neural network to extract key features as a function of time, finding the trends of the sensor data over time. Because this type of data invariably has a lot of noise, we also leveraged a max pool function that served to smooth the data. This approach required treating the data as an "image," with sensor data arranged by timestamp. Since TuringSense calculates joint angle data, each of six sensors outputs three data points. Thus, the first three rows of the "data image" contain the data for the first sensor, the second three contain the data for the second, and so on. With a convolutional neural network, applying a filter allowed the network to look at the same sensor data over time rather than treating all of the data as a one dimensional input as in

our previous models. This model applies two convolutions, first with a 3x3 filter to look at three timestamps of a particular sensor, then with a 6x5 filter. This also includes 10 layers, which was experimentally determined by trial and error by looking at accuracy values in the dev and train sets. After the two convolutions, we use a max pooling layer, which essentially smooths out variations in the data. In testing, we attempted to use an average pooling layer, which ultimately converged to the same accuracy but which required more epochs and a larger window. Thus, we opted to use max pool because it seemed to minimize computational needs overall. After max pooling, we flatten the data and run it through a fully connected layer with a softmax activation. This ultimately outputs a vector of five outputs with predictions, which we then compare to the label data of one-hot vectors. We had attempted to add another convolution layer, but this greatly decreased accuracy regardless of what hyperparameters and filter sizes we chose, so we ultimately ended up with the aforementioned architecture. The sensors are fairly accurate, and the general trend of the data appears consistent within stroke types, so it seems to make sense that a simpler model can adequately address the data. We again use cross entropy loss to compare the output predictions with the labels.

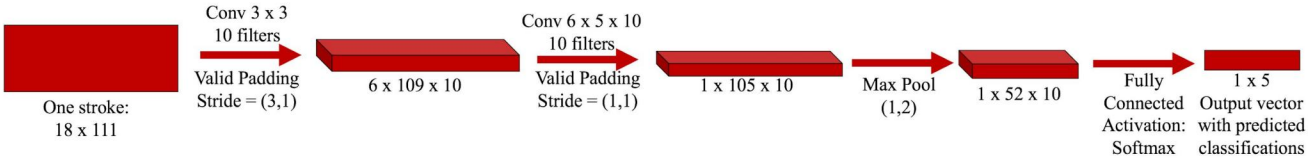


Figure 4. Our final convolutional neural network model.

Results and Discussion

In testing, the CNN outperformed both other architectures. A 3 x 3 convolution proved to be more effective than an 18 x 3 convolution, which made sense because the former case treated each sensor as disparate components while the latter considered all the sensor data simultaneously. We chose the learning rate and the epochs through trial and error, starting large at 0.1 and ultimately settling on 0.001 as the optimum. After 55 epochs, the accuracy of the dev set stopped increasing even as the training accuracy continued, which indicated that we were overfitting the data. We then tried changing the window of the max pool and attempted to use average pool instead, however both accuracies ultimately converged to the same value, 0.98, so we ultimately chose max pool with a window of (1,2). After setting into the number of epochs, learning rate, and overall architecture, we tried changing the number of filters. The accuracy dropped off after 10 filters in the first layer, but continued to improve as we approached 10 from below. Simultaneously, we attempted different values for the second layer filters, and the accuracy was always the best when the number of second layer filters were the the same or nearly the same as the first layer. This makes sense because since we are using valid padding, the convolution decreases the dimensions in itself, and with fewer filters, the total data points for the next layer are diminished significantly.

	Best Train Accuracy	Best Dev Accuracy	Best Test Accuracy
1 Hidden Layer	0.9754	0.94	0.96
4 Hidden Layers	0.9862	0.96	0.97
CNN	0.9938	0.98	0.98

Table 1. Performances for three attempted architectures. These are the best results we were able to obtain for each model.

Overall, the average accuracy of our model was 98%. A caveat to remember is the small sample set of our data, the initial weight optimizations had a large impact on the overall results, and the only way we were able to tune our hyperparameters meant that we needed to use a random seed to start off in each run.

The several errors made throughout testing were spread out across different stroke types, so each stroke type classification had more or less the same number of correctly predicted shots. When analyzing where the model made mistakes, it seems that the model had difficulty recognizing misshapen data (either a bad shot or sensor error).

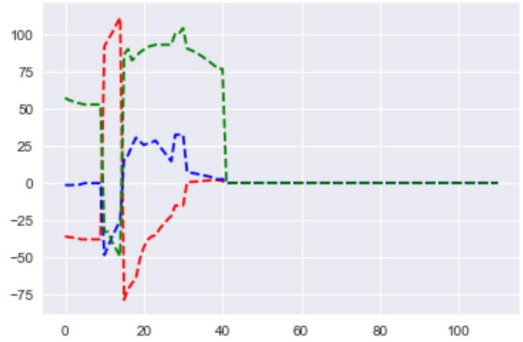


Figure 5. This is data from sensor 2 of stroke type 1 (forehand flat). In this case, there was an anomaly in the sensor.

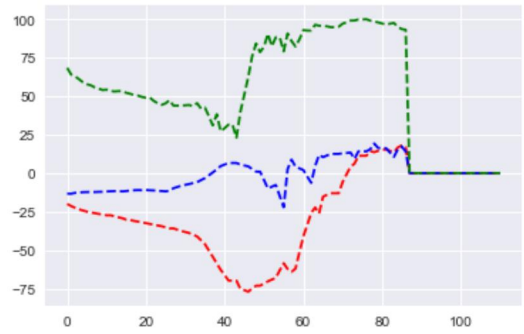


Figure 6. An accurate reading from sensor 2 of stroke type 1 (forehand flat).

Because there was little variability in data between the training set and the test/dev set, there did not seem to be much overfitting as the dev set had near identical accuracies with the test set.

Overall, the CNN architecture was able to adapt to changes in the standard deviation of the noise we added, suggesting its worth as the best model for which to proceed. Naturally, real-life data will have greater variability, and thus a model which is most adaptable to variability in data while looking for trends with respect to time is the best.

Conclusion/Future Work

All three models we tested had very high accuracy given the amount of data we had. While this constitutes success, and while we were able to get very high accuracy on our train, dev, and test sets, we are aware that our lack of data severely constrains our ability to refine this project. For one, we have only been working with one specific professional tennis player to collect our data, which is good for ensuring that each stroke is accurate and is done with the right technique. However, we have not trained the model on amateurs, where there would be more variability in strokes.

Looking forward, we plan to continue this project this summer, focusing on several things. One is to increase training set size and variability to include both professional players and amateur players. With variability in players, the model would also adapt to extract deterministic features of the stroke. Different players have different playstyles, and as a final analysis we looked at the only batch performed by a different human the model trained on.

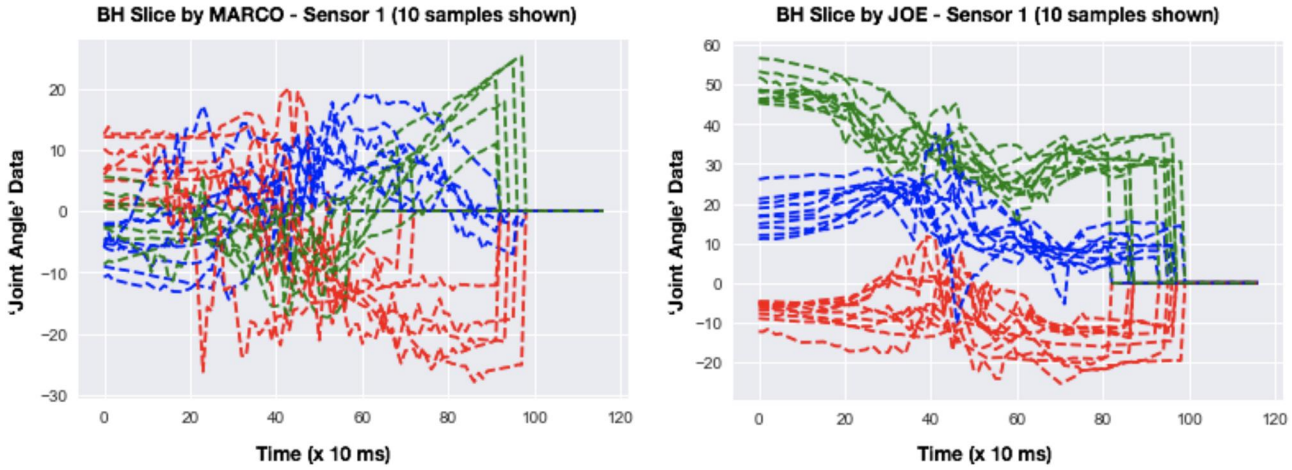


Figure 7. 10 stroke samples of a backhand slice by two different players, which show considerable variability, especially at the start.

It appears that different players may start the stroke differently, suggesting that the deterministic factor of a certain stroke is towards the end. Further, it would make sense to implement better cleaning methods that discard strokes not within the bounds of a reasonable stroke time(50-100ms, more statistical analysis needed). Interesting results may also arise if the data is scaled so that all the strokes are represented over the same time span as opposed to having to pad most strokes with zeros. Overall, this project demonstrated the predictability of stroke types with six TuringSense sensors.

GitHub Repository link: https://github.com/ero123/CS-230-Project

Contributions

Ellen Roper wrote, implemented, and tested all of the code for the single layer network, the deep four layer network, and the convolutional neural network. She conducted all of the model training, the hyperparameter tuning, and did the debugging for all three as well. She also the code for the data parsing, augmentation, and formatting, and she created the presentation session poster. Ellen also wrote half of the project write up, edited and formatted the report.

Tristan Gosakti conceptualized, wrote and implemented the code for data refining (including data parsing, cleaning, padding and formatting). He learned to build graphs/plots and used statistical tools to optimize the input size, reach insights about data like variability in stroke, sensor anomalies, distribution of time each stroke takes, etc. These were used to help advise designing the architecture of the CNN, deciding how to augment data and deciding how to distribute the data into different sets for testing. He also wrote half of the project write up and edited/finalized the whole report to consolidate all key concepts covered throughout the project. He initiated the opportunity to work with Turingsense.

References

Connaghan, Damien, et al. "Multi-Sensor Classification of Tennis Strokes." *2011 IEEE SENSORS Proceedings*, 2011, doi:10.1109/icsens.2011.6127084.

Jiang, Wenchao, and Zhaozheng Yin. "Human Activity Recognition Using Wearable Sensors by Deep Convolutional Neural Networks." Proceedings of the 23rd ACM International Conference on Multimedia - MM '15, 26 Oct. 2015, pp. 1307–1310., doi:10.1145/2733373.2806333.

Ronao, Charissa Ann, and Sung-Bae Cho. "Human Activity Recognition with Smartphone Sensors Using Deep Learning Neural Networks." Expert Systems with Applications, vol. 59, 15 Oct. 2016, pp. 235–244., doi:10.1016/j.eswa.2016.04.032.

Shah, Hitesh, et al. "Automated Stroke Classification in Tennis." *Lecture Notes in Computer Science Image Analysis and Recognition*, pp. 1128–1137., doi:10.1007/978-3-540-74260-9_100.

TensorFlow. (2018). Get Started| TensorFlow. https://www.tensorflow.org/get_started/

Acknowledgments

Many thanks to our TA Patrick Cho, who gave us advice in trying different architectures. We were going to try an LSTM network, but found that a CNN made more sense despite it being an architecture commonly used for image processing – ironically the field that TuringSense is aiming to upend.