# CS229 Project Final Report:
# Basketball Player Motion Classification
*Category: Computer Vision*

Ray Xie rayxie11@stanford.edu
Jiaxuan Su suj98@stanford.edu
Tingkang Wang wangtk@stanford.edu

## 1 Introduction

Keeping track of players' motion during basketball matches is of great strategic importance. It can not only help coaches analyze players' behavioral patterns, but also serve to identify the tendency of potential fouls in advance. In this project, we want to use deep learning to classify players' motion into several categories: walk, run, defense, pass, shoot etc. We believe the existence of such an algorithm can facilitate downstream applications such as AI referees and play breakdowns of the opposing team. This project utilizes an online dataset [1] of labeled short videos (16 frames each) capturing one motion of a single player in each clip and the coordinates of the player's joint. The first frame of the video is used as the input to the initial model while the joint coordinates in every frame of each video are used as the inputs to the improved model. The initial model used two fully connected layers to output the softmax prediction of the possibilities of each motion, while the improved model used recurrent neural networks (RNN) with a bi-directional long-short term memory (LSTM) layer to output the softmax prediction of the possibilities of each motion.

## 2 Related Work

Basketball player motion classification can be abstracted to a more general topic: human body motion classification. There is a dichotomy in this field: studying with images and videos or skeleton data produced by sensors worn on human bodies. The advantage of images and videos is the ease of acquisition (online videos or videos shot from a smartphone) while the disadvantage is also apparent: no depth perception of joint positions and difficulty in feature extraction. The advantage of wearable sensor data is the inclusion of depth in joint motion. Acquisition of these data will be hard since it requires test subjects to wear these sensors while in motion. Both methods have been explored in depth.

Skeleton data is usually in 5 dimensional space. It is important to transform these data to a readable form for machine learning models. One method is to transform skeleton data into color maps and use CNN to learn from these maps [2]. This method reached an average accuracy of 86% for motions like walking, kicking and pushing. However, not all joints are important in different body motions. The addition of an attention model [3] which applies different importance factors to each skeleton point has proven to perform better (an average accuracy of 94%) than only CNN.

For motion classification using photos and videos, extracting human motion features is the most important. The prevalent method is to mark human joints (pose estimation) at each frame of the video. A simple CNN can get the job done but there are other challenges [4]. One that concerns this project the most is missing joints: either blocked by other body parts or out of frame. Unsupervised learning using an overcomplete auto-encoder can lower the rate of missing joints in large datasets by almost threefolds [5]. After obtaining joint coordinates from each frame of the video, time series classification methods are used in the classification of body motion in which ROCKET achieved the best accuracy [6].

In this project, we draw inspiration from motion classification methods using photos and videos. The team decided to approach the task in two different ways. First, a simple fully connected model is used to learn from the first frame of all videos (images). Second, joint coordinates for each frame of the video is learned using the LSTM bidirectional RNN model which is one kind of time series model.

# 3 Dataset and Features

## 3.1 Dataset description

As mentioned in the previous section, the dataset used in this project is composed of two different inputs: 16 frame videos of athletes conducting a single motion and the respective joint coordinates in each frame. The labels for the dataset are numbered in the following way: 0 Block, 1 Pass, 2 Run, 3 Dribble, 4 Shoot, 5 Ball in Hand, 6 Defense, 7 Pick, 8 no_action, 9 Walk, 10 Discard. The dataset contains a total of 37085 of correctly labeled examples. The ratio between training set and test set is 2:1. During model parameter tuning using validation sets, the team didn't achieve better results. That's why there is no validation set.

## 3.2 Data pre-processing

Predicting a specific number has proven to be hard on Machine Learning models. Thus, the team has decided to convert the above number labels into one-hot labels of length 11. For example, the "Pass" motion will be converted from 1 to [0,1,0,0,0,0,0,0,0,0,0]. As for inputs, the group has decided to break them down in different ways. For the videos, the team extracted the first frame of videos. For the time evolution of joint coordinates, the team flattened the joint coordinates in each frame.

## 3.3 Extraction of first frame

Extraction of the first frame is straight forward. The group used Python's cv2 package [7] to extract the values from the first frame as three dimensional matrices: the first two dimensions specify the location of each pixel and the third dimension contains the RGB values of each pixel. In addition, the group decided to use two different training methods. The first one is to further modify the extracted frame. Taking the average of each pixel at the third dimension would give us a grayscale image of the frame. Then, spread the remaining 2 dimensions of the matrix into a 1 dimensional vector.

## 3.4 Flattening of joint coordinates

Flattening of joint coordinates is a little tricky. The dataset already has the joint coordinates of each frame from the videos packaged as dictionaries. Each dictionary has joint indices as keys and tuples of coordinates as values. Since there are 16 frames in each video, there are 16 dictionaries for each video. There are a total of 18 joints (indexed from 0 to 17) in the dataset. However, due to the 2 dimensional nature of videos in general, some joints are blocked by the torso or other moving joints. There are occasions where some joints are missing from single frames (dictionary length shorter than 18). There are also cases where a particular joint is absent from all frames. The group has devised three ways to counteract this issue. The first one is to fill absent joint coordinates with (0,0). The second method is to take the mean of a specific joint in every frame and fill the frames where the joint is absent with that mean. If a joint is absent from all frames, it will be filled with (0,0). The third method is to linearly interpolate the coordinates of a specific joint in all frames and fill the frames where the joint is absent with the interpolated results. If a joint is absent from all frames, it will be filled with (0,0). The last method is to pad the missing values with the existing values in other frames. Again, if a joint is absent from all frames, it will be filled with (0,0). All of these processing methods are done with the help of Python's Pandas package [8].

# 4 Methods

## 4.1 Baseline model

The initial model proposed in this project is a purely fully connected model with two layers. Considering that nonlinearity could exist among pixels, the first layer is implemented with two most commonly used activation functions, 'sigmoid' and 'ReLU'. The second layer is implemented using the softmax function. Loss is calculated by averaging the cross entropy loss on the training batch using one-hot labels. The following equations show the forward propagation for a single input using this two-layer model:

$$a = \sigma(W^1 x + b^1)$$
$$z = W^2 a + b^2$$
$$y = softmax(z)$$

Where σ is the sigmoid function or the ReLU function.

For the initial model, only the greyscale first frame of the video clips is used as input to the model. The first frame is extracted, converted to grayscale and reshaped using the method explained in the previous section. The model is trained using mini batch gradient descent with cross entropy loss calculated using one-hot labels.

### 4.2 RNN model

To take advantage of the temporal dimension of data, we implemented an RNN model. We used a bi-directional LSTM layer in our model. LSTM stands for "long-short term memory". It is a kind of RNN layer that can store time series information and retrieve important features despite gap length.

The model first performs a batch normalization on the input. The normalized data then goes through the bi-directional LSTM layer. Next, the output of the LSTM layer is regularized by a dropout layer and put into two fully-connected layers. Finally, the category of motion is output by a softmax activation function. The network architecture is given by the following figure.

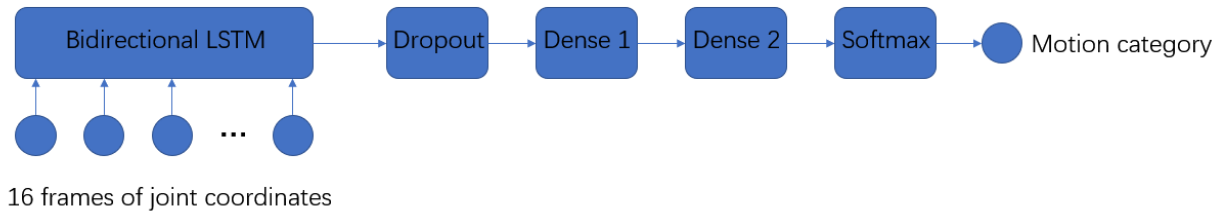


Figure 1. RNN model architecture.

## 5 Experiments/Results/Discussion

### 5.1 Softmax regression model

For the initial fully connected model, the same 'train', 'dev' and 'test' sets are used across all the experiments. The 'train' set contains 12500 videos, while the 'dev' set contains 2500 videos and the 'test' set contains 1000 videos. The experiment was first run with the sigmoid first layer with a layer size of 300. The model was trained for 30 epochs with a learning rate of 2 and a mini batch size of 250. The model was experimented both with and without regularization. The following figures show the training and testing loss and accuracy for the non-regularized and regularized models:

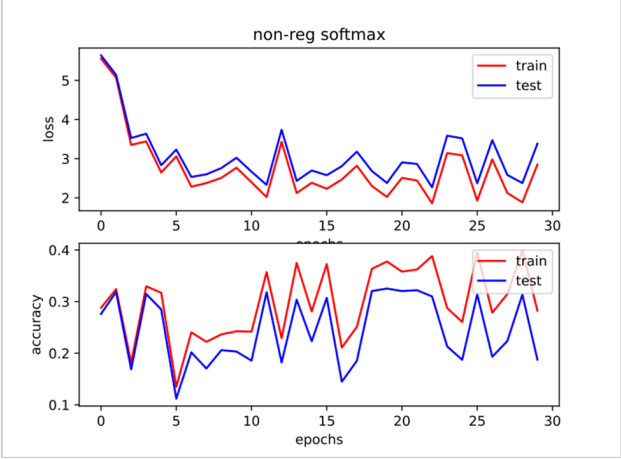


Figure 2. Non-regularized softmax model

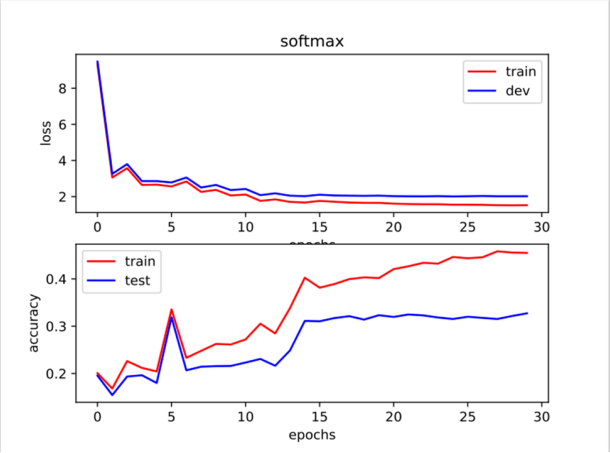


Figure 3. Regularized softmax model

As shown in the figures, some unusual behavior was observed. Though the general trend of the loss and accuracy was reasonable, they also tended to go up and down between epochs especially for the non-regularized model. One of the

possible reasons was the learning rate being too big. Also during the training process, a few overflows were encountered in the exponential functions of the sigmoid function. After investigating the size of the input vector, one of the possible causes was determined to be the input size being too large. This was reasonable considering that the image contained more than 20000 pixels. After computing the dot product with the weight matrix, it was easy for the resulting value to reach the overflow limit of Numpy exponential function which is around 709. To deal with the overflow issue, a small trick was done to the sigmoid function. After computing the dot product of the weight matrix and the input vector, the values were divided by 100 before adding the bias to keep them below the limit. The following figures show the training and testing loss and accuracy for both the non-regularized and regularized models after implementing the small trick for preventing overflow:

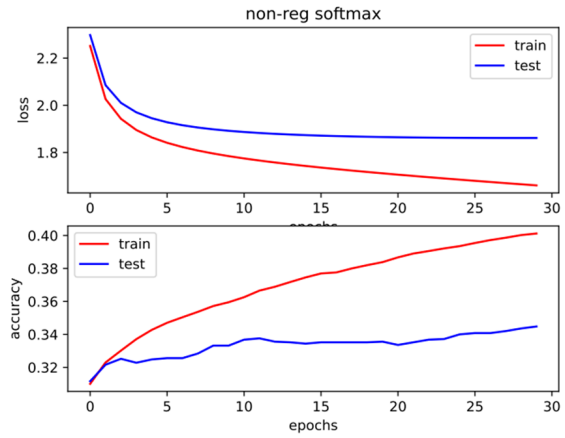


Figure 4. Non-regularized softmax model

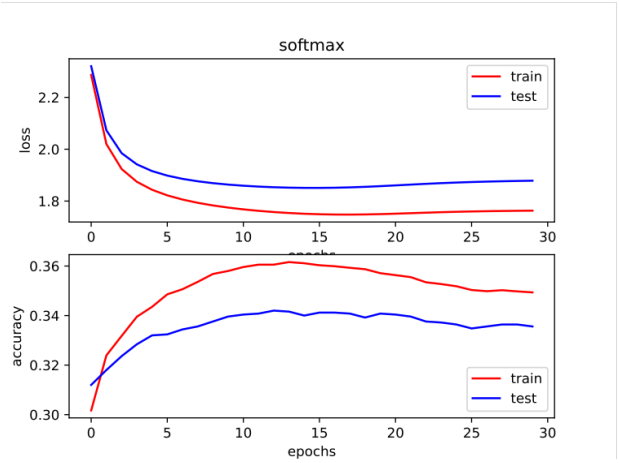


Figure 5. Regularized softmax model

The results show that the trick was effective. The model was trained effectively. As shown in figure 3, the performance difference between the training set and test set is huge. This proves that regularization is needed to prevent overfitting. As shown in figure 4, with regularization, the performance gap between the training set and the test set became smaller, but the accuracy started to go down after training for around 13 epochs. This might be caused by too much regularization and this is left for further experiments. ReLU function with regularization was also tested. While it performed worse on the training set than the non-regularized softmax model, it had similar performance on the test set, which means the performance gap was smaller.

### 5.2 RNN model

The LSTM layer of the RNN model was first designed to be unidirectional. Such property made the model capture fewer features during players' motion and result in poorer performance than the bidirectional one. Its best accuracy and loss curves are shown below. The best test accuracy turns out to be 65.8%.

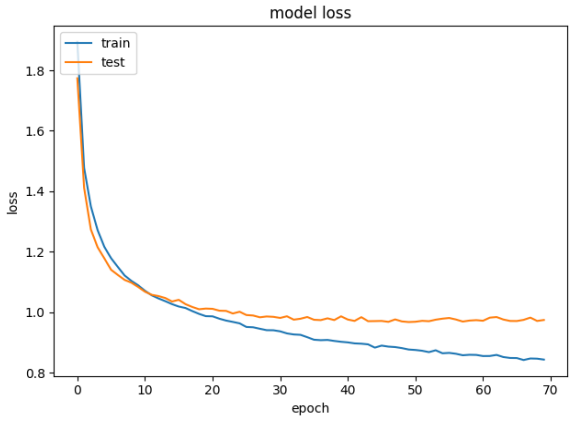


Figure 6. Unidirectional RNN model loss

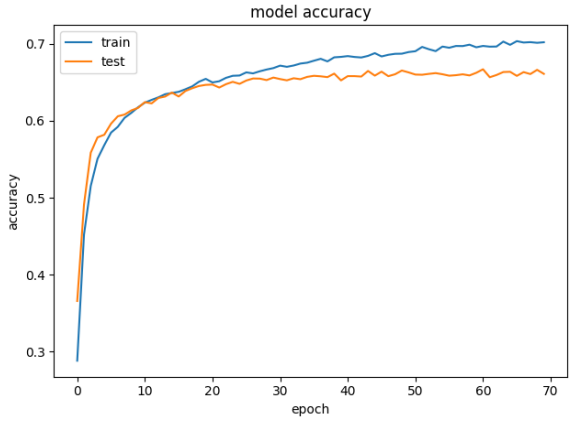


Figure 7. Unidirectional RNN model accuracy

We then changed the LSTM layer to be bidirectional. To tune the model hyperparameters, we first impose some ranges on them. We decided to choose LSTM neuron number from {20, 30, 40, 50}; dropout rate from {0.2, 0.4, 0.6}; hidden unit number from {11, 20, 30} and epochs from {50, 60, 70, 80, 90, 100}. Notice that there were also dropout layers designed for dense layers, but the best dropout rate turned out to be 0. The best combination is determined by accuracy on the test set. The tuned hyperparameters are given in the following table.

Table 1. Hyperparameters for the tuned RNN model.

| | |
|---|---|
| Bi LSTM | neurons: 30 |
| dropout | dropout rate: 0.2 |
| Dense 1, 2 | hidden units: 11; activation: relu |
| Compilation | optimizer: Adam; epochs: 70; batch size: 128 |

The following figures show the training and test loss and accuracy for the RNN model. The training loss is categorical cross entropy loss and the training metric is accuracy.

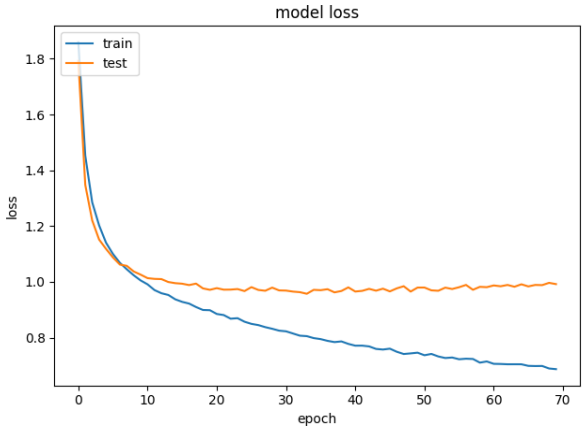


Figure 8. Bidirectional RNN model loss

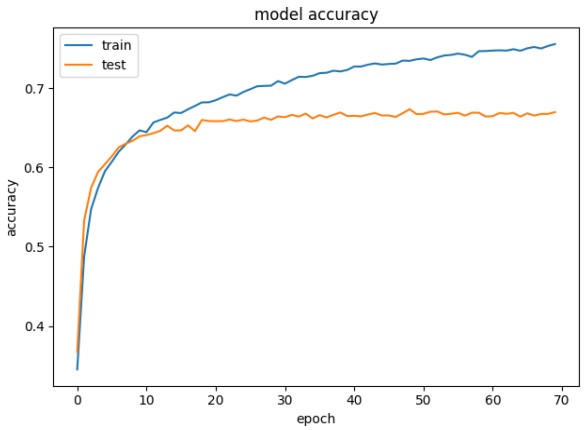


Figure 9. Bidirectional RNN model accuracy

The test accuracy of the final tuned RNN model reaches 67.7%, which is the highest among all models.

# 6 Conclusion/Future Work

In this report, we presented two models to categorize basketball players' motion. The softmax regression model serves as a baseline model and has an accuracy of 40%. We then designed a RNN model and tried out different architectures and hyperparameters. The final accuracy is given by 67.7%. Taking advantage of the temporal characteristic of the data, the RNN model achieved a better accuracy.

If given more time, we first want to expand our dataset. As is shown in the accuracy curve above, overfit remains a big issue in our dataset despite various regularization has been tried out. Having more data will enable us to train the model with less overfit and thus have better performance. Moreover, we would like to also train a CNN model to automatically label joint coordinates for frames in videos. Such a model combined with our RNN model will create an end-to-end application of directly categorizing motion from videos.

# 7 Contributions

Ray Xie: Data collection and preprocessing, research on related works, design neural network architecture
Jiaxuan Su: Data preprocessing, design neural network architecture, model implementation
Tingkang Wang: Data preprocessing, design neural network architecture, model implementation.

# 8 References


[1]. Francia, S. (2020). *SpaceJam: A dataset for Basketball Action Recognition*. GitHub. Retrieved May 6, 2022, from https://github.com/simonefrancia/SpaceJam

[2]. Liu, M., Liu, H., & Chen, C. (2017). Enhanced skeleton visualization for view invariant human action recognition. Pattern Recognition, 68, 346–362. https://doi.org/10.1016/j.patcog.2017.02.030

[3]. Zhu, K., Wang, R., Zhao, Q., Cheng, J., & Tao, D. (2019). A cuboid CNN model with an attention mechanism for skeleton-based action recognition. *IEEE Transactions on Multimedia*, *22*(11), 2977–2989. https://doi.org/10.1109/tmm.2019.2962304

[4]. Cronin, N. J. (2021). Using deep neural networks for kinematic analysis: Challenges and opportunities. *Journal of Biomechanics*, *123*, 110460. https://doi.org/10.1016/j.jbiomech.2021.110460

[5]. Carissimi, N., Rota, P., Beyan, C., & Murino, V. (2019). Filling the gaps: Predicting missing joints of human poses using denoising autoencoders. *Lecture Notes in Computer Science*, 364–379. https://doi.org/10.1007/978-3-030-11012-3_29

[6]. Singh, A., Le, B. T., Nguyen, T. L., Whelan, D., O'Reilly, M., Caulfield, B., & Ifrim, G. (2022). Interpretable classification of human exercise videos through pose estimation and multivariate time series analysis. *AI for Disease Surveillance and Pandemic Intelligence*, 181–199. https://doi.org/10.1007/978-3-030-93080-6_14

[7]. *Opencv-python*. PyPI. (n.d.). Retrieved June 6, 2022, from https://pypi.org/project/opencv-python/

[8]. *Pandas*. pandas. (n.d.). Retrieved June 6, 2022, from https://pandas.pydata.org/