Exercise Transcription with CTC Loss

Evan Sabri Eyuboglu
Computer Science
Stanford University
Email: eyuboglu@stanford.edu

Geoffrey Angus
Computer Science
Stanford University
Email: gangus@stanford.edu

Roozbeh Mahdavian
Computer Science
Stanford University
Email: rooz@stanford.edu

Pierce Freeman
Computer Science
Stanford University
Email: piercef@stanford.edu

***Abstract*—We present a supervised learning method for exercise recognition and transcription. Our algorithm ingests micro-electromagnetic (MEM) sequences recorded by a wearable device during a workout set and outputs both the type of exercise performed and the number of repetitions completed. Our model consists of a 1-D convolutional layer whose output is fed into a Bidirectional Long Short-Term Memory Recurrent Neural Network (BLSTM). Because exercise transcription is a many-to-many sequence prediction problem where input and output sequences are unaligned and different in length, we use Connectionist Temporal Classification (CTC) Loss to train the network. Our model generalizes to new users not included in the training set and achieves near-perfect accuracy in transcribing their workouts.**

Keywords*—*Convolutional Neural Networks, Bidirectional Long Short-Term Memory, Connectionist Temporal Classification, Wearable Technology.

I. Introduction

Many wearable devices contain Micro Electro-Mechanical (MEM) sensors, tiny accelerometers and gyroscopes embedded into the device that collect data describing the device's movement. Existing algorithms that interpret this MEM data can recognize a diverse set of gestures and human activities with remarkable accuracy [8].

Avid weight-lifters and physical therapy patients often write down their sets to monitor their progress and stay on top of their exercise regimens. Manually recording workouts at the gym is imprecise and can interrupt the flow of a workout. To keep track of a physical therapy prescription, physical therapists usually rely on exercise logs taken by their patients. However, patients could make mistakes when recording exercises or they might exaggerate or misrepresent how closely they are adhering to the prescription. A wearable device capable of accurately transcribing exercises could serve as a simple, high-fidelity method of exercise regimen logging.

Here we present a deep-learning model that accurately transcribes weight-lifting workouts. Our model accepts as input a time-series of MEM data recorded during the workout and outputs a transcription of the exercises performed. The model distinguishes between four weight-lifting exercises: dumbbell curl, barbell bench press, power clean and back squat.

There are two main challenges inherent to applying supervised learning to exercise transcription: (1) training a learning model demands large-scale data collection from many weight-lifters and (2) any MEM time-series data we do collect is unaligned— that is, the beginning and end of each repetition in a set is not annotated.

(1) To make data collection scalable we've developed an Apple Watch app that records MEM data for a lift and allows the user to easily label the MEM sequence with the type and number of repetitions performed. (2) Furthermore, we've implemented a deep-learning architecture to overcome the alignment problem. The architecture was inspired by work in Natural Language Processing to overcome a similar alignment challenge. It consists of a 1-D convolutional layer whose output is fed into Bidirectional Long Short-Term Memory Recurrent Neural Network (BLSTM). We call this a Long-Term Recurrent Convolutional Network, or LRCN, as this is a version of the model architecture pioneered by Donahue *et al.*, which uses a Convolutional Neural Network as a feature extractor for a Long Short-Term Memory Recurrent Neural Network responsible for sequence prediction. [2] Because exercise transcription is a many-to-many sequence prediction problem where input and output sequences differ in length and are unaligned, we use Connectionist Temporal Classification (CTC) Loss to train the network.

Our model achieves near-perfect accuracy in transcribing the workouts in our dataset. Most importantly, the model generalizes to new users not included in the training set.

II. Related Literature

Existing methods for exercise and gesture recognition can be grouped into five general categories: (1) temporal warping models, (2) vanilla machine learning models that rely on substantial feature engineering, (3) Hidden Markov Models (HMM), (4) convolutional neural networks, and (5) recurrent neural networks. (1) The *uWave* algorithm developed by Liu *et al.* is representative of temporal warping models [9]. Temporal warping models like *uWave* work remarkably well for small datasets, but perform worse than other models as the scale of the training dataset increases. (2) Several other studies have designed intricate feature engineering processes that take raw MEM data as input and build informative feature vectors that are then fed into simple machine learning models (support vector machines and feed-forward neural networks) [4]. For example, Manini *et al.*'s approach extracts feature vectors encoding frequency and intensity from the raw MEM signal and feeds those vectors into a support vector machine classifier. (3) Since MEM data is naturally sequential in nature, Hidden Markov Models have long been a popular approach. Hoffman *et al.* presented the seminal paper on HMM applied to gesture recognition [7]. In their and most other models, an HMM is trained for each activity or gesture. (4) Duffner *et al.* recently presented a powerful convolutional neural network architecture for gesture recognition [3]. Finally, (5) Lefebvre *et*

Fig. 1. The data-collection process on our RepKit Apple Watch app.

al. developed a recurrent neural network model that leveraged BLSTM cells to accurately label gestures [8].

The above works inspire our model in a variety of ways, some more direct than others. However, in most of these studies, the training data was aligned. Few studies in exercise and gesture recognition have tackled unaligned activity recognition, as we hope to do.

III. Problem Formulation

The task at hand is to predict the exercise sequence performed during a given MEM input sequence. This problem can be framed as a many-to-many sequence prediction task where input and output sequences are unaligned.

Formally, an input MEM sequence $x^{(i)}$ can be represented by an $n \times t^{(i)}$ matrix where n is the dimensionality of the input sequence and $t^{(i)}$ is the length of the sequence. The Apple Watch outputs 12 MEM readings so $n = 12$. The output sequence $y^{(i)}$ can be represented by a $\ell^{(i)}$ dimensional vector of exercise labels drawn from some set of exercises E. Our goal is to predict some sequence $\hat{y}^{(i)}$ that matches $y^{(i)}$.

$$\text{Input: } x^{(i)} \in \mathbb{R}^{n \times t^{(i)}}, \text{ Output: } \hat{y}^{(i)} \in E^{\ell^{(i)}} \tag{1}$$

IV. Data Collection

Our goal was to design a deep learning model that could run on today's wearables. This goal necessitated a dataset that accurately reflected in-production technology. We chose to focus on the Apple Watch since it has the largest market share within the space. [5] There are no existing datasets with the sample rate and feature data required for our task.

To collect data from both our research team and colleagues, we focused on building a scaleable pipeline to record and label data. The pipeline built consists of three phases:

A. watchOS app

The RepKit watchOS app allows participants to quickly record and label exercises and their corresponding repetitions (Figure 2). A participant first selects an activity to record. Once recording, RepKit engages both the accelerometer and gyroscope, which samples processed motion data at 100Hz. Each processed motion unit is a 12-dimensional vector which provides the attitude, torque, gravity, and acceleration (each of which are 3-dimensional vectors over an x, y, and z axis).

Attitude provides the orientation of the device (as pitch, roll, and yaw) with respect to a given reference frame. Torque provides the rotational acceleration of the device with respect to the device frame (i.e., along the device's own axis). Gravity provides the gravitational acceleration of the device with respect to the device frame, and acceleration provides the positional acceleration of the device with respect to the device frame.

All of the accumulated motion data is kept in-memory until a participant taps finish, at which point they can label it with the number of repetitions. After saving the set, the number of sets is updated and displayed on the application's Active Workout view. After a participant has recorded a satisfactory number of sets, the participant presses the save button and all of the unsychronized exercise archives are then transferred to the iOS app.

B. iOS app

watchOS syncs with a counterpart iOS application to aggregate data into a more permanent medium. Once on device, users can also choose to disregard a certain recorded set by viewing the annotation and deleting it. Once data has been validated, this information is then transmitted to a centralized web-application.

A phone equipped with GPS also allows us to use location services to identify when users are close to a supported gymnasium. During data collection, this feature allowed us to "gameify" our application and reminded users to annotate their workouts.

C. Central database

In order to accumulate data from multiple mobile devices, we built a centralized repository for all our device data. The application built supports standard REST operations to collect all the motion data and labels recorded on the watch application.

This method also allows for dynamic integration with our training infrastructure. Via a GET request, our Tensorflow training code can download the most recent training examples and add these to our train/dev/test sets. We archive the data locally on-device to make sure that we are performing evaluation on a static dataset.

D. Dataset Split

Of the six weight-lifters that participated in data collection, five were included in the train set. The dev and test sets were made up of the examples from the last participant. We explicitly excluded the last participant from the train set. The exact makeup of our dataset can be found in Fig. 2. We chose this dataset split so that our experiments would best reflect how an exercise transcription model might actually be deployed in a real world setting. Presumably, the model would not have access to training data for new users. Furthermore, this dataset split allows us to confirm that our model is learning patterns inherent to weight-lifting exercises, not just memorizing participant idiosyncrasies.

Dataset Split		
Train	Dev	Test
1606 examples	249 examples	249 examples
5 participants	1 unseen, random participant	

Fig. 2. The size and makeup of our three dataset splits.

The train set was used to train the weights of the model, the dev set used to tune hyperparameters and assess different architecture, and our test set was used exactly once at the end to assess the performance of our best model.

E. Re-rack Labels

Initially, our output sequences consisted only of the exercises performed. However, when collecting data, the watch would also record participants re-racking weights after a set. Failure analysis on preliminary models performed by plotting BLSTM softmax outputs against MEM input sequences showed that our model often attempted to classify the "re-rack" motion as one of the 4 exercise classes. This led to high levels of avoidable bias in our preliminary experiments. We combated this issue by adding 4 exercise-dependent "re-rack" labels to the model's class vocabulary and rebuilding our dataset such that each output sequence contained a new "re-rack" label associated with its exercise type. As an example, this transformed output sequences from ["CURL", "CURL", "CURL"] to ["CURL", "CURL", "CURL", "RERACK_CURL"]. By doing this, we increased the expressiveness of our model and substantially decreased avoidable bias.

V. Model Architecture

Our final model architecture combined a shallow 1-D Convolutional Neural Network (CNN) with a Bidirectional Long Short-Term Memory Recurrent Neural Network (BLSTM) optimized over a Connectionist Temporal Classification (CTC) loss function [6].

A. Bidirectional Long Short-Term Memory

As discussed above, the MEM input sequence can be represented as a time-series matrix $x^{(i)} \in \mathbb{R}^{n \times t^{(i)}}$ where n is the dimensionality of the input sequence and $t^{(i)}$ is the length of the sequence. Long Short-Term Memory Recurrent Neural Networks (LSTM) are capable of capturing long-term dependencies in time-series data and are, thus, well-suited for modeling MEM sequence patterns [1]. We realized that the final "re-rack" motion of free-weight exercises contains a lot of information about the nature of the exercise itself. This led us to opt out of vanilla LSTMs in favor of Bidirectional LSTMs (BLSTMs). At each time-step, BLSTMs can use future information as well as past information to inform its predictions. Our BLSTM (Fig. 3.iii) outputs a softmax probability distribution over the possible exercise labels at each time-step. Specifically, at each time-step t and example (i) the BLSTM output is:

$$p_t(a_t^{(i)}|x^{(i)}) \quad \forall a_t^{(i)} \in \hat{E} \tag{2}$$

where $a_t^{(i)} \in \hat{E}$ is some exercise label. Note that $a_t^{(i)}$ is drawn from $\hat{E} = E \cup \{\epsilon\}$, adding the ϵ label is necessary for applying

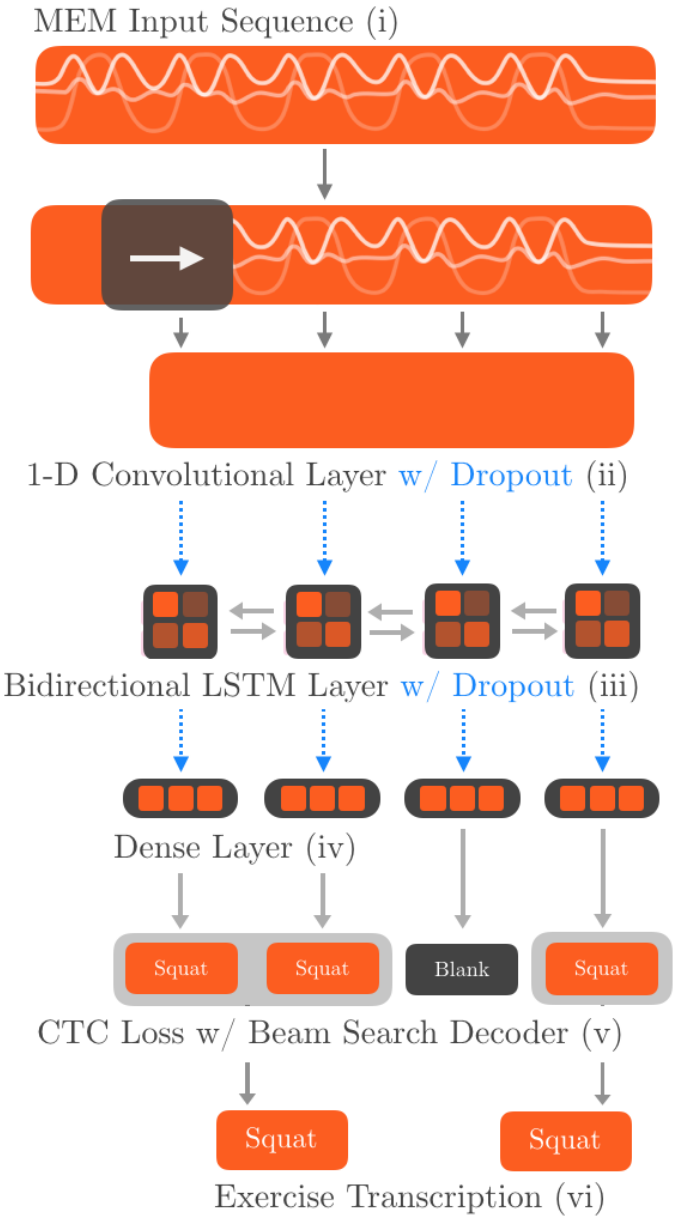

Fig. 3. The model architecture.

CTC loss, which we describe below. Fig. 6 shows the BLSTM softmax output for each time-frame.

B. Connectionist Temporal Classification

To train a BLSTM-RNN we must define some loss function over our training dataset. As discussed above, the input sequences $x^{(i)}$ and output sequences $y^{(i)}$ in our training set are unaligned and differ greatly in length. Thus, defining a loss function for the model is non-trivial: it would not suffice to define a simple cross-entropy loss term for the softmax output at each time-frame, since the correct output for that specific time-frame is unknown, even for labeled examples.

Connectionist Temporal Classification (CTC) loss is often used in speech recognition tasks where input and output sequences are unaligned [6]. CTC loss deals with different length input and output sequences by introducing the idea of

a valid alignment. A valid alignment for an output sequence $y^{(i)} \in E^{\ell^{(i)}}$ is a sequence of labels $a^{(i)} \in \hat{E}^{t^{(i)}}$ that when collapsed produce the sequence $y^{(i)}$. To collapse an alignment, we simply combine all adjacent labels of the same type and remove ϵ labels. This is shown above in Fig. 3.v. Given the softmax output of the BLSTM from above, the probability of an alignment is given by the following equation:

$$p(a^{(i)}|x^{(i)}) = \prod_{t=1}^{t^{(i)}} p(a_t^{(i)}|x^{(i)}) \tag{3}$$

For a given output $y^{(i)}$, we can compute the set of all valid alignments of length $t^{(i)}$ which we denote $A^{(i)}$. Given the softmax outputs of the BLSTM, we can then compute the probability of the given output by marginalizing over all possible alignments:

$$p(y^{(i)}|x^{(i)}) = \sum_{a^{(i)} \in A^{(i)}} p(a^{(i)}|x^{(i)}) \tag{4}$$

We can now compute the probability that our model assigns for the true label $y^{(i)}$. To train our model, we seek to maximize the likelihood of observing our training set given the input sequence $x^{(i)}$. With this intuition, we can define the following loss function:

$$\ell(\text{model}) = \sum_{i=1}^{m} -\log p(y^{(i)}|x^{(i)}) \tag{5}$$

The loss function above can be efficiently computed using dynamic programming. To perform inference, we can decode the softmax outputs using a beam search over all alignments of length $t^{(i)}$.

In the preliminary stages of our research, we attempted to train a simple BLSTM using the CTC loss function. This proved unsuccessful because our input MEM sequences were sampled at 100 Hz and were, thus, too long for the BLSTM to handle. We resolved this issue using a variety of downsampling techniques.

C. Downsampling and Convolutional Layers

Our baseline model, a simple BLSTM using the CTC loss function, was wholly ineffective at predicting both exercise type and count. We learned that its failures were due to the high-sample rate associated with each of the input sequences. For many of the input sequences, the number of timesteps were of the order of one thousand. As it turns out, LSTMs— though conceived to solve this very issue— are limited in their ability to learn long-term dependencies in sequences of this length. We experimented with several downsampling techniques and saw drastically improved results.

Our first set of experiments utilized a naïve downsampling technique that, for each $x^{(i)}$, simply reduced the dimensionality of input sequence $x^{(i)}$ by replacing it with a new sequence $\widetilde{x}^{(i)}$ that consisted of every kth timestep of x, where k was a tuneable hyperparameter. Naïve downsampling proved to be very effective, achieving high accuracy upon reducing t for each input sequence by a factor of 20.

We then experimented with rebinning the time sequences. Given that n is the number of features and t is the number of timesteps for some sequence, we reshaped each $n \times t$ sequence to be of shape $n \times \frac{t}{k}$, where k was some tunable hyper parameter. We used neighborhood averaging in the process of rebinning the data in order to minimize the information lost by the preprocessing step. This approach was less effective than the naïve approach, possibly due to the smooth and thus less expressive nature of the newly processed input sequences.

Our final attempt at reducing the timestep dimensionality of each input sequence relied not on preprocessing, but on a fundamental change in our model architecture. We added a 1-D Convolutional Layer to the model (see Fig. 3.ii. The convolutional layer accepts $x^{(i)}$ as input and outputs directly to the BLSTM. While this does not exactly downsample the data, it does change the dimensionality of each sequence from $n \times t$ to $c \times \frac{(t-f)}{s} + 1$, where c, f and s are the number of filters, the filter width and length of stride, respectively. This was by far the most effective method of reducing timestep dimensionality as it both compressed the data comprehensively and added expressiveness to the model. The convolutional layer is able to shrink the input sequence to a reasonable length, while also interpreting the details offered by high frequency sampling.

D. Full Architecture

At prediction time, our end-to-end model accepts an MEM sequence as input (Fig. 3.i) and feeds it through a 1-D convolutional layer with large stride (Fig. 3.ii). The condensed sequence emitted by the convolutional layer is fed into a BLSTM recurrent neural network (Fig. 3.iii). The outputs of the BLSTM are passed through a dense layer with softmax output (Fig. 3.iv). Finally, these softmax outputs are decoded using a CTC beam search decoder (Fig. 3.v) producing an exercise transcription (Fig. 3.vi).

VI. Experiments & Results

A. Model and Hyperparameter Experiments

We performed an iterative hyperparameter search. On each iteration we chose hyperparameters at random within some range, and then narrowed our range around the hyperparameter values that yielded the best results. Finally, once we narrowed the range of hyperparameter values sufficiently, we performed a small grid search. In total we ran twenty-five experiments to

Model Accuracy: Set Transcription			
Model Type	Train Accuracy	Dev Accuracy	Training Time
BLSTM-50 (without re-rack)	77.2%	73.0%	38 min
BLSTM-50 (with re-rack)	84.2%	78.3%	60 min
BLSTM-150 (with re-rack)	95.8%	96.9%	225 min
CONV-64 BLSTM-128 (with re-rack)	99.5%	97.5%	36 min

Fig. 4. The exercise recognition and set transcription accuracy of four model versions. The addition of a "re-rack" label significantly increased accuracy in the dev set, as did the addition of more hidden units.

tune our model. For each experiment, we built our model with specific hyperparameters, trained it on the train set and evaluated it on the dev set. We assessed each model by measuring its set transcription accuracy, which is computed by counting the fraction of examples that are accurately transcribed. An example is accurately transcribed if the model reports both the correct exercise and the correct number of repetitions.

In Fig. 4, we've outlined some key results from our model and hyperparameter search. Model performance benefited greatly from the addition of the "re-rack" label. As we discussed above, the decision to add the "re-rack" label was motivated by a failure analysis that involved plotting BLSTM output against the input MEM sequence. An example of such a plot is given in Fig. 6. After adding the "re-rack" label, we sought to strike a balance between model complexity (number of filters and hidden units) and regularization strength (dropout rate) in order to improve the performance of the model. Ultimately we found a sweet-spot that eliminated avoidable bias while also avoiding overfitting. Here it is worth noting that including a convolutional layer before our BLSTM layer both increased our accuracy and dramatically sped up training time, reducing the training time of the model from 3 hours and 45 minutes to 36 minutes.

In our final model we use a dropout rate of 10%, batch size of 16 and learning rate of 0.001. The convolutional layer consists of 64 filters each of size 50. Our stride is of length 20. The BLSTM contains 128 hidden units and 1 hidden layer. It is also worth noting that dropout in the BLSTM was applied only to input and output channels, not the state channel. All of these parameters were chosen via our hyperparameter search.

B. Results

After experimentation, we deemed the Conv-64 BLSTM-128 model using "re-rack" labels to be our most robust model. As seen in Fig. 5, the model exhibited both low bias and relatively low variance, achieving a 99.5% set transcription accuracy on the training set and a 97.5% set transcription accuracy on the dev set. **The model achieved a test set transcription accuracy of 98.4%.**

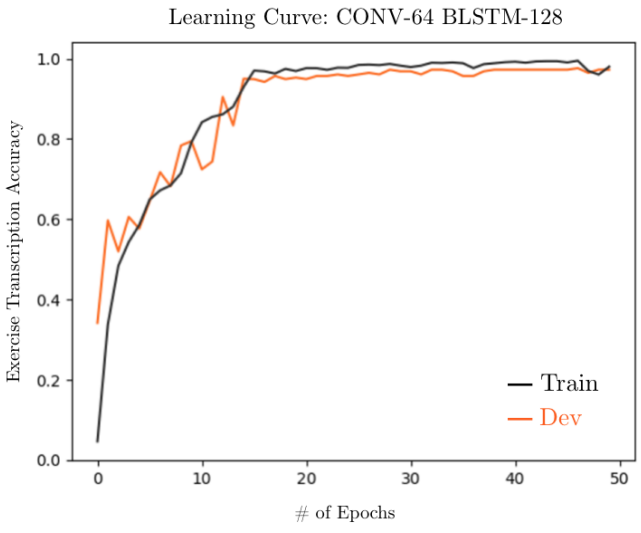

Fig. 5. The learning curve of the CONV-64 BLSTM-128 model on both the training set and the dev set. The vertical axis is the transcription accuracy. Correctness is defined as the identification of both the correct exercise type and the repetition count.

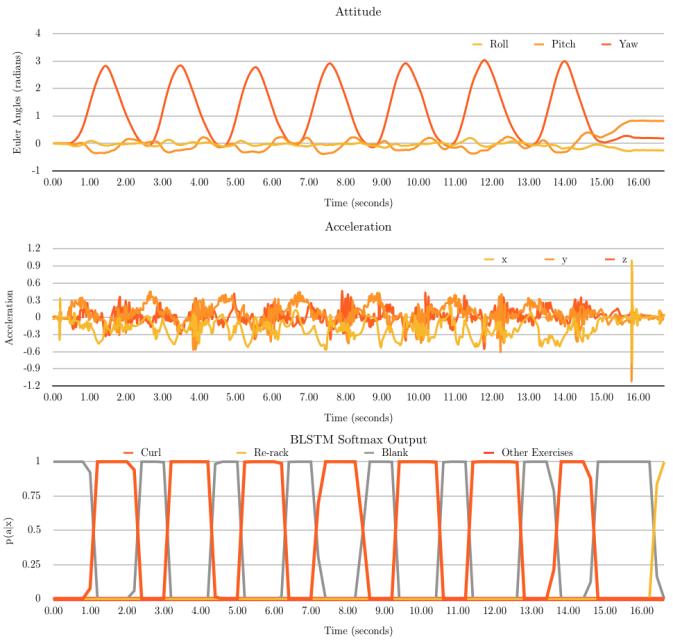

Fig. 6. The softmax output of the BLSTM layer aligned with the MEM sequence input on a 7 curl exercise. Note that the softmax output of the class Re-rack approaches 1 at the end of the workout.

In Fig. 6 we plot the BLSTM softmax output on a seven curl exercise. High-probability output for the "curl" label are correctly correlated with the yaw oscillation. Further, one can observe that the "re-rack" label associated with the curl exercise approaches 1 at the end of the sequence as desired. A closer examination of the input sequences reveal similar relationships between other features and the softmax output of each exercise class. A similar plot can be produced for all four exercise types. During failure analysis, we discovered that many of the errors made by the model were likely caused by mislabeled data.

VII. Conclusion & Future Work

We set out to apply deep learning methods to the task of exercise transcription and we found with near-perfect results. The most promising aspect of our research was the model's ability to generalize its learning to unseen weightlifters. Our model thus has the potential to provide value in real world applications.

We must run more tests in order to confirm its efficacy on the unseen weightlifter. Upon completing this task, the brunt of the future work will be in expanding the functionality of the model. As it stands, the model does not detect the beginnings of workouts; it is only capable of classifying the end of recorded exercises. To resolve this, we would build a simple model capable of solving the similar speech recognition "trigger word detection" task. Creating a two-model pipeline would allow future users to move in and out of workouts without ever turning off the program.

The model's near-perfect accuracy on unseen data is evidence of the effectiveness of LRCNs on MEM sequence data. The model shows great promise and lays a foundation for future exercise transcription systems.

VIII. Contribution

All members contributed to all parts of the project. Geoff and Sabri focused on building the model architecture, tuning hyperparameters, running experiments, the poster and the final writeup. Pierce and Rooz focused on developing the Apple Watch app, iOS app, Heroku database, and dataset pipeline. All members contributed to the data collection effort.

IX. Implementation

The implementation of the dataset collection system, the model and the experiment harness is available at: `https://github.com/piercefreeman/RepKit`

References

[1] S. Hochreiter and J. Urgen Schmidhuber, "LONG SHORT-TERM MEMORY," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: http://www7.informatik.tu-muenchen.de/~hochreit%20http://www.idsia.ch/~juergen.

[2] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description," Nov. 2014. [Online]. Available: http://arxiv.org/abs/1411.4389.

[3] S. Duffner, S. Berlemont, G. Lefebvre, and C. Garcia, "3D gesture classification with convolutional neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, May 2014, pp. 5432–5436, ISBN: 978-1-4799-2893-4. DOI: 10.1109/ICASSP.2014.6854641. [Online]. Available: http://ieeexplore.ieee.org/document/6854641/.

[4] R. Xie and J. Cao, "Accelerometer-Based Hand Gesture Recognition by Neural Network and Similarity Matching," *IEEE Sensors Journal*, 2016, ISSN: 1530437X. DOI: 10.1109/JSEN.2016.2546942.

[5] Anita Balakrishnan, *IDC data: Apple Watch on top of wearables market*, 2018. [Online]. Available: https://www.cnbc.com/2018/03/01/idc-data-apple-watch-on-top-of-wearables-market.html.

[6] A. Graves, A. Ch, S. Fernández, S. Ch, F. Gomez, T. Ch, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," [Online]. Available: https://www.cs.toronto.edu/~graves/icml_2006.pdf.

[7] F. G. Hofmann, P. Heyer, and G. Hommel, "Velocity Profile Based Recognition of Dynamic Gestures with Discrete Hidden Markov Models," [Online]. Available: https://link.springer.com/content/pdf/10.1007%2FBFb0052991.pdf.

[8] G. Lefebvre, S. Berlemont, F. Mamalet, and C. Garcia, "BLSTM-RNN based 3D Gesture Classification," [Online]. Available: https://hal.archives-ouvertes.fr/hal-01224806/document.

[9] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan, "uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications,"