IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 24, 2019, accepted September 4, 2019, date of publication September 12, 2019,
date of current version September 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2940701

Simulating Time-Series Data for Improved Deep Neural Network Performance

JORDAN YEOMANS[1], SIMON THWAITES[2], WILLIAM S. P. ROBERTSON[3], DAVID BOOTH[4], BRIAN NG[5], AND DOMINIC THEWLIS[2]

[1]Australian Institute for Machine Learning, The University of Adelaide, SA 5000, Australia
[2]Adelaide Medical School, The University of Adelaide, SA 5000, Australia
[3]School of Mechanical Engineering, The University of Adelaide, SA 5000, Australia
[4]Defence Science and Technology Group, Adelaide 5111, Australia
[5]School of Electrical and Electronic Engineering, The University of Adelaide, SA 5000, Australia

Corresponding author: Dominic Thewlis (dominic.thewlis@adelaide.edu.au)

This work was supported in part by the University of Adelaide Interdisciplinary Research Fund and Defence Science and Technology Group. The work of D. Thewlis was supported by a National Health and Medical Research Council Career Development Fellowship (ID:1126229).

ABSTRACT Deep learning algorithms have shown remarkable performance in classification tasks, however, they typically perform poorly with small training datasets due to overfitting. Overfitting occurs for all data types, although for the purposes of this study we are interested in time-based signals. This study introduces a novel technique to simulate time series signals from a dataset of categorically labeled data which can be used to train a deep neural network. The objective is to improve the predictive accuracy of a deep neural network on a separate validation dataset. To demonstrate the simulation methodology and improvements to the model's performance, a small dataset of ground reaction forces was used with the goal of identifying a person based on the raw signal. Our results show that the simulation method presented improves validation accuracy and reduces model training time for each of the three signal types.

INDEX TERMS Deep learning, deep neural networks, data simulation, data augmentation, time-series classification, time-series data augmentation, transfer learning, LSTM, 1D CNN, ground reaction force, personal identification, small dataset, overfitting.

I. INTRODUCTION

Deep learning algorithms typically perform poorly with small training datasets [1]. One area we are especially interested in is improving model performance on time-series data particularly when the datasets are limited. Long Short Term Memory (LSTM) networks are widely regarded as the gold standard of deep learning algorithms to analyze time-series data [2]. However, they are prone to overfitting [3], which significantly limits validation performance. Overfitting occurs when the model learns extremely specific relationships within the training data that are not present in the validation data [1]. In situations where the training dataset is small, the likelihood the model will overfit increases. The number of training examples needed to build a model that is able to generalize without overfitting depends on model architecture, model complexity, the number of categories and the nature of the data. Many techniques exist to reduce overfitting [4]; however, research has shown that a solution to model overfitting is often as simple as increasing the size of the training dataset [5]. However, the size of the training dataset can be limited by many factors from budget to availability of data (e.g. it is not possible to collect more stock prices than those already available). In the area of biomechanics, machine learning has become popular in recent years. The continual limitation in this field of research is the acquisition of sufficient data. This was highlighted by Haliaj et al. [6] who noted that most studies using machine learning were limited by sample size and hence susceptible to overfitting.

Transfer learning is a technique to reduce overfitting and has been used in studies with limited data for image classification tasks. Esteva et al. [7] showed expert human level competency could be achieved using transfer learning to categories skin cancer from a set of 129,450 images. These techniques utilize a pre-trained image classification model, however, these types of pre-trained models do not exist for time-series data.

An alternate method to reduce overfitting is to simulate data for training. For example, Kim et al. [8] developed a simulated room including virtual sound sources and reflective

The associate editor coordinating the review of this manuscript and approving it for publication was Ting Li.

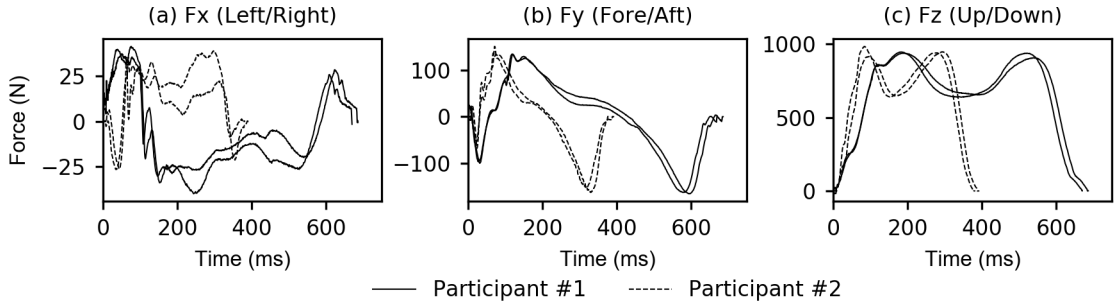

FIGURE 1. Examples of two GRF signals for two participants from a dataset. (a) F_x is the lateral/medial force. (b) F_y is the anterior/posterior force. (c) F_z is the vertical force. Note that the data shown in figure 1 are not the same as the data used in figure 2, i.e. these figures are based on data from different people.

walls to generate utterances to train a neural network. The simulation method was shown to substantially improve performance. Data augmentation is perhaps the most familiar and frequently used approach for reducing model over fitting. The following section provides and overview of current approaches used and their limitations.

A. DATA AUGMENTATION

Within deep learning frameworks, one approach for dealing with model overfitting is data augmentation. This array of techniques has shown success in computer vision problems [9]. Image based data augmentation techniques includes image-resizing, image-rotation, left-right flipping, and zooming. The purpose of data augmentation is to increase the size of the training set with images that are different to the original set, yet the new images still accurately represent the class/label. Within the time-series classification domain, various techniques have been proposed. Each method slightly changes the input signal data in such a way that it can still be considered as a new input signal, without causing it to no longer represent the original label.

Salamon et al. described a variety of approaches such as time stretching and pitch stretching. These techniques slightly modify the input by scaling the signal in the time (without changing the pitch), and pitch domain respectively. In each case, the classification of audio signals was improved. Adding background noise was also used, which can reduce overfitting to noise in the signal [10]. Guennec et al. used two methods called window-slicing and window-warping. Window-slicing is based on a concept from computer vision where portions of the signal are trained on. During test time, multiple slices re-size random sections of the signal [11].

Um et al. analysed a variety of data augmentation techniques when monitoring Parkinson's Disease using a wearable device. Jittering, scaling, rotation, permutation, MagWarp, TimeWarp and Cropping were used individually, and in combination with each other, during the training of a CNN. It was found that for this task signal rotation lead to the greatest single improvement, likely due to the variation that occurs when the sensor is placed on the person's body [12].

Khandakar et al. recently showed that by combining four data augmentation techniques they could improve the performance of a time-series classification task when using a deep neural network. An Inertial Measurement Unit recorded signals from an excavator, and by using a LSTM they classified the type of task being performed. Four techniques, namely jittering, scaling, rotation and time-warping were combined, which substantially improved the model's accuracy [13].

Collectively this body of work indicates that different augmentation techniques have varying results for different datasets. This suggests that not all data-augmentation techniques work equally well for all classification tasks, or for different datasets. As such, adding new data-augmentation techniques to the toolbox, which can be tested and used on a case-by-case basis by the community is a valuable contribution to the field.

II. PURPOSE

This paper presents a novel technique to improve the accuracy of deep learning models for time-series datasets with limited data. The work does not aim to demonstrate superiority of our approach to data augmentation to alternative approaches. We do not present any comparisons at this stage as the level of detail required would distract from the novel method presented. Rather, we aimed to show that this method is suitable for a particular use-case, and suggest that it should be added to the set of data-augmented strategies currently available. This paper has been divided into two sections:

1) *a method to simulate time-series training data from a small initial dataset*; and
2) *an experimental demonstration that the simulated dataset can improve a deep learning model.*

To demonstrate the simulation method we have chosen to use a time-series dataset common to the area of biomechanics: the ground reaction force (GRF) signal. Our underlying hypothesis is that the GRF, which is the force measured as people walk, could be used as a method of personal identification [14] using deep neural networks.

III. SIMULATION METHODS

Examples of the raw signals used for this method are presented in Fig. 1. To describe the simulation process we used the F_x signal (Fig. 1(a)) The simulation method developed has been designed for categorically organized

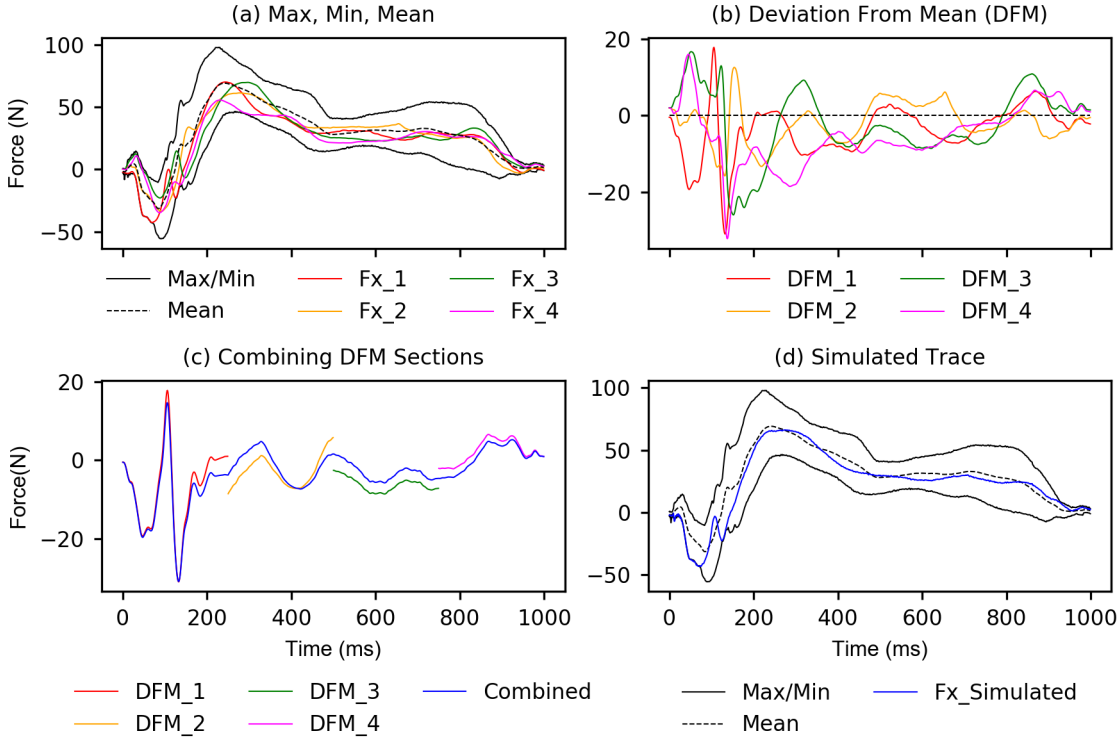

FIGURE 2. (a) Shown in colour are four random signals from a particular category. In this case it shows four different F_x traces from our dataset. These signals could be any four signals for a particular category in the dataset to be simulated. The calculated min/max bounding box (black solid lines) and mean curve (black dashed line) are also shown. (b) The deviation from the mean (DFM) for the four signals used in the simulation. (c) Randomly selected sections from the DFM curves are selected and scaled (coloured lines). They are then combined to generate a surrogate DFM (blue line). (d) The completed simulated signal is shown in blue.

1-dimensional (1-D) signals. It is a requirement for the method that more than one signal per category exist. Signals were first resized to a standard length of 1000 time steps, which is modifiable based on the dataset. Once resized the simulation process for a chosen category, or for a participant in this case, is as follows:

1) Signals were smoothed with a Savitzky-Golay filter [15] (in this work, SciPy's `savgol_filter` was used [16]). A record of each signal's noise was retained by taking the difference of the original and smoothed signal. This step was required to remove the noise from the average waveform, which is essential for Step 4.
2) Where values were <0, all category were offset signals to ensure all values ≥ 0. All signals in the category were offset by the same amount. This was essential to prevent negative values being multiplied together in Step 7 and then reversed in Step 8.
3) Calculate the maximum (max) and minimum (min) value of all of the category's smoothed signals for all time steps to generate a bounding curve (Fig. 2(a), black solid lines). Note 1: Fig. 2(a) shows four example signals for this category (coloured lines). Note 2: the F_x1 trace defined the min bounding curve for the first 75ms (Fig. 2(a), red line).
4) Calculate the mean curve as the mean of the bounding curve for all time steps (Fig. 2(a), black dashed line).
5) Calculate the deviation from the mean (DFM) curve for all of the category's smoothed signals (Fig. 2(b), colored lines).
6) Generate a surrogate DFM curve by joining randomly selected true DFM sections. Sections were scaled to meet the next section at the mean of their difference (Fig. 2(c), blue line). Multiple sections from the same category were used. The number of sections was changed between simulations to introduce variability. To scale, a linear gradient was applied to the section to ensure start and end points end up where needed.
7) The simulated curve was computed as the surrogate DFM multiplied by the mean curve.
8) Where the signals were originally offset for Step 2), the offset was subtracted from the simulated curve.
9) Finally, randomly chosen noise signal from the category was added to the simulated curve.

The final simulated signal is shown in Fig. 2(d), blue line.

IV. EXPERIMENTAL DEMONSTRATION

Ground reaction force data from a cohort of 79 participants were collected in a controlled laboratory environment for two sessions at least three days apart. On each occasion, participants were instructed to walk the length of a laboratory (14m) walking over two force platforms (AMTI OPT400600HF,

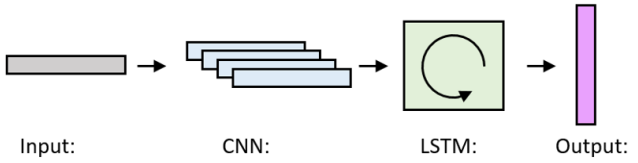

FIGURE 3. Deep learning model used in the experimental demonstration.

AMTI, US) recording the GRFs at 2000 Hz (Fig. 1). The data were then resampled to 1000Hz.

Session 1 traces were used as the basis for simulating new data, which were used for neural network training. Session 2 was used exclusively for model validation. Between all participants, 2076 (≈26.3 examples per participant) GRF traces were recorded for Session 1 and 2053 for Session 2 (≈25.9 examples per participant).

The deep learning model used was a 9 layer 1-D convolutional neural network (CNN) $->$ 3 layer LSTM $->$ one-hot categorical prediction (Fig. 3). The model did not use any final fully connected (FC) layers, which is relatively uncommon for image classification tasks [17] but has shown success for LSTM-based models [18] The CNN was built using three patterns of: 3x CNN layers (kernel $= 5$) $->$ 1x max pooling (stride $= 2$, pool $= 2$) $->$ 1x batch normalization. For each pattern (P) the CNN filters (f) were constant. P1: f $= 32$; P2: f $= 64$; P3: f $= 128$. CNN layers were initialized with truncated normalized values (mean $= 0$, std. dev. $=$ 0.1) and used L2 regularization. ReLu activation functions between CNN layers were used. LSTM layers had 128 hidden units per layer with a tanh activation functions. No regularization was used and all units were initialized to zero. The last vector of values from the LSTM was the input to the final prediction layer.

A Tensorflow implementation can be seen in the Appendix. A batch size of 512 and a learning rate of 0.0003 was used and no dropout was included. All input signals were globally normalized to be between 0-1. Tensorflow's Adam optimizer, and Softmax Cross Entropy with Logits V2 loss function have been used.

Training was completed using The University of Adelaide's supercomputer, Phoenix. Each model was trained using one NVIDIA Tesla K80 GPU (CUDA v9.0 and cuDNN v7.3). Each model was trained for 11 hours to compare training time performance.

V. METHODS TO MEASURE PERFORMANCE INCREASE

Results were calculated for two scenarios to allow for direct comparison:

1) *Case 1: the baseline case was trained on 2076 real Session 1 signals and validated on 2053 real Session 2 signals.*
2) *Case 2: the augmented method was trained on 500,000 simulated signal (using the approach described in section III) and validated on 2053 real Session 2 signals.*

TABLE 1. Training time performance.

Force	Case 1 accuracy	Case 2 accuracy	Relative improvement	Error reduction
F_x	35.2%	71.0%	101.7%	55.2%
F_y	52.6%	77.6%	47.5%	52.7%
F_z	60.2%	79.4%	31.9%	48.2%
$F_x+F_y+F_z$	69.4%	86.1%	24.1%	54.6%

The prediction accuracy, which is the percentage of correct predictions over all validation signals, was calculated using the validation set (Session 2). It is important to note the signals used for validation have not been used in the simulation or training process.

The prediction accuracy for each signal was calculated and is shown in Table 1 as F_x (lateral/medial), F_y (anterior/posterior), and F_z (superior/inferior). An ensemble prediction was calculated as the sum of the softmax predictions for F_x, F_y, and F_z. The participant with the highest summated prediction was used as the predicted participant. This is shown as $F_x + F_y + F_z$ in Table 1.

Two metrics were used to evaluate the accuracy performance of the method.

1) Relative Improvement: the change in predictive accuracy (acc) compared to the baseline case:

$$\text{improv}_{\text{rel}} = \frac{\text{acc}_{\text{sim}}}{\text{acc}_{\text{base}}} - 1 \tag{1}$$

2) Error (e) reduction: the percentage of the original error that the method has reduced:

$$e_{\text{reduce}} = \frac{\Delta e}{e_{\text{base}}} = \frac{(1 - \text{acc}_{\text{base}}) - (1 - \text{acc}_{\text{sim}})}{(1 - \text{acc}_{\text{base}})} \tag{2}$$

where acc_sim is the validation accuracy achieved on Session 2 when trained on simulation data and acc_base is the accuracy achieved on Session 2 when trained on unchanged (real) Session 1 data.

To quantify the improvement in training time an additional two metrics have been calculated:

1) Time for each model to reach 95% of maximum validation accuracy.
2) Time for each model to reach 95% of the maximum accuracy achieved by the baseline method (Case 1).

For each metric, the percentage reduction was calculated as:

$$e_{\text{reduce}} = \frac{\Delta\text{time}}{\text{time}_{\text{base}}} \tag{3}$$

where Δtime was the difference in the time taken to reach 95% of respective model accuracy used for comparison.

VI. RESULTS

The simulation method improved performance for all signals. Summary data are presented in Table 1.

The validation accuracy for the F_x signal has improved from 35.2% (Case 1) to 71.0% (Case 2) representing

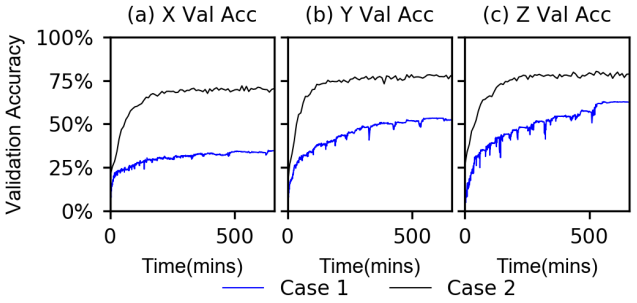

FIGURE 4. Comparison of validation accuracy improvement over time for each model. Blue shows models trained on real data (Case 1). Black shows models trained on simulated data (Case 2). All models have been trained on the same hardware (NVIDIA Tesla K80 GPU).

TABLE 2. Training time performance.

	Metric 1: Time to reach 95% of each model's accuracy			Metric 2 Time to reach 95% of Case 1 accuracy		
Force	Case 1 (min)	Case 2 (min)	Reduce (%)	Case 1 (min)	Case 2 (min)	Reduce (%)
F_x	480	208	56.7%	480	27	94.4%
F_y	371	163	56.1%	371	45	87.9%
F_z	526	199	62.2%	526	63	88.0%

a relative improvement of 101.7% and an error reduction of 55.2%. F_y has improved from 52.6% to 77.6% which is a relative improvement of 47.5% and error reduction of 52.7%. F_z improved from 60.2% to 79.4% giving a relative improvement of 31.9% and error reduction of 48.2%. The ensemble model improved from 69.4% to 86.1%, relative improvement of 24.1% and error reduction of 54.6%.

The validation accuracy over time is shown in Fig. 4 and results are tabulated in Table 2. The time for the F_x Case 1 to reach 95% of its own peak accuracy is 480 min compared to 208 min (Case 2), a reduction in self compared training time of 56.7%. Comparing to Case 1, the simulation (Case 2) achieves the 95% peak accuracy achieved in Case 1 in 27 min, a 94.4% reduction.

The training time for F_y Case 1 was 371 min compared to 208 min (Case 2) a reduction of 56.1%. Case 2 achieves the same results of Case 1 in 45 min, a reduction of 87.9%. The training time for F_z Case 1 is 526 min compared to 63 min (Case 2) a reduction of 62.2%. Case 2 achieves the same results of Case 1 in 63 min, a reduction of 88.0%.

VII. DISCUSSION

This study presents a new method for simulating categorically labeled time-series data from a small dataset to reduce the effects of overfitting when using deep learning algorithms. The method was designed to be somewhat data agnostic (in the context of time-series data). In the context of a specific example, we have shown that this approach can improve both classification performance and training time for three unique GRF signals in an experimental demonstration.

As an intuition as to why the method improves the training of a neural network: the features within the data, some of which are unique to the category, are being augmented into slightly new orientations. Although a category may have unique features consistent between recorded data it is unlikely that they will occur in an identical manner (e.g. magnitude/time). Augmenting the data using this method allows the network to learn the features that represent the category, across a variety of orientations and magnitudes.

For each of the three signals, the simulation method reduced the error in the range of 48.2% to 55.2%. The relative accuracy improvement ranged from 31.9% to 101.7%. The most significant relative improvement was F_x, it is thought this could be for two reasons: 1) the F_x signal is more variable than either the F_y or F_z signals meaning the model trained with simulated data benefited from the data augmentation the most; and 2) the original accuracy was the lowest of the three signals, and hence had the most room to improve; in addition to small percentage increases resulting in a large relative increase.

The improvements in validation accuracy brought about by this method will be beneficial for models limited by small datasets that require a higher accuracy. The training time for each of the signals has also improved. The time taken for the model to train to 95% of its own peak accuracy has reduced in the range of 56.7% to 62.2%. When calculating the time for the simulation model to reach 95% of the baseline case the reduced training time is in the range of 87.9% to 94.4%. Reducing the training time for deep learning algorithms is always beneficial. Additionally, it is imagined this would be extremely beneficial in cases where a model is achieving the desired accuracy but requires intermittent retraining. It is possible the improvements to training time could extend to large datasets as well.

The model used in this study was a combination of a 1-D CNN and LSTM networks. Within the computer vision space it has been shown CNN layers act as feature detectors. LSTM networks, on the other hand, are designed to find temporal dependencies within a dataset. Using a combination of these architectures has resulted in the CNN layers learning features within the signal and the LSTM layers then finding the temporal relationship between those features to provide a prediction. It is expected the simulation method is helping both portions of the network. To our knowledge this is the first time this method of simulating time series data has been published. Methods of data augmentation exist but tend to augment an individual signal rather than combining signals within a single category. Methods also exist for simulating data, however, they remain domain specific.

A. LIMITATIONS

It is acknowledged this work has some limitations which require further research. Firstly, this method will not suit all time-series datasets. A requirement of the method is that signals must to be able to separated into distinct categories, so that signals from the same category can be combined into a simulated signal. The method does assume that all samples within a category have a similar temporal spacing/pattern, although different signal lengths can be resized as shown in this work. In other words, the general shape of each signal needs to be similar. It is untested if simulating a combination of differently shaped signals could still be beneficial. The method has not been tested on data outside of the GRF domain and we have not researched the sensitivity to the number of samples per category. Secondly, we have not compared the method described in this paper to alternative data augmentation techniques. The goal of this work was to describe and evaluate the performance of this method, not to compare data augmentation methods. However, future work should look to address this limitation, with the different time-series signals. Thirdly, this approach required the signals to be re-sized to a standard length. This could be removing important information about category. This step could be reversed after the simulation, though this was not tested.

VIII. CONCLUSION

In this study, we have demonstrated a new method for simulating data from a small, labeled dataset of time-series data to improve the performance of deep learning algorithms. The method developed is general in nature. We have shown that the method improves validation accuracy and reduced training time.

APPENDIX-NEURAL NETWORK IMPLEMENTATION

```
# Tensorflow Version >= 1.10

  # — Input/Output Data —
  # train_input_data = [#_of_samples, signal_length,
    #_of_channels]
  # train_output_data = [#_of_samples, #_of_
    categories_as_1_hot_array]

  # — Neural Network Parameters —
  lstm_hidden_units = 128
  batch_size = 512
  learning_rate = 0.0003
  init = tf.initializers.truncated_normal(stddev=0.1)
  regularizer = tf.nn.l2_loss

  # — Create Model Placeholders —
  x = tf.placeholder(tf.float32, shape=(None,
    train_input_data.shape[1], train_input_data.shape[2]))
  y = tf.placeholder(tf.float32, shape
    =(None, train_output_data.shape[1]))

  # — Create 1D CNN Network —
  nn = tf.layers.conv1d(x, filters=32, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.conv1d(nn, filters=32, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.conv1d(nn, filters=32, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.max_pooling1d(nn, strides=2,
     pool_size =2)
  nn = tf.layers.batch_normalization(nn)

  nn = tf.layers.conv1d(nn, filters=64, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.conv1d(nn, filters=64, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.conv1d(nn, filters=64, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.max_pooling1d(nn, strides
     =2, pool_size=2)
  nn = tf.layers.batch_normalization(nn)

  nn = tf.layers.conv1d(nn, filters=128, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.conv1d(nn, filters=128, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.conv1d(nn, filters=128, kernel_size=5,
     activation=tf.nn.relu, kernel_initializer=init,
     kernel_regularizer=regularizer)
  nn = tf.layers.max_pooling1d(nn, strides
     =2, pool_size=2)
  nn = tf.layers.batch_normalization(nn)

  # — Create LSTM Network —
  cell_1 = tf.nn.rnn_cell.LSTMCell(lstm_hidden_units)
  cell_2 = tf.nn.rnn_cell.LSTMCell(lstm_hidden_units)
  cell_3 = tf.nn.rnn_cell.LSTMCell(lstm_hidden_units)
  multicell = tf.nn.rnn_cell.MultiRNNCell([cell_1,
     cell_2, cell_3])

  nn, state = tf.nn.dynamic_rnn(multicell, nn, dtype
     =tf.float32)
  nn = tf.transpose(nn, [1, 0, 2])
  nn = tf.gather(nn, int(nn.get_shape()[0]) - 1)

  # — Create Fully Connected Layer —
  nn = tf.layers.dense(nn, train_output_data.shape[1],
     activation=None)
```

```
# — Create Loss —
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_
    with_logits_v2(logits=nn, labels=train_output_data))

# — Create Optimizer —
optimizer = tf.train.AdamOptimizer(learning_rate
    =learning_rate).minimize(loss)
```

ACKNOWLEDGMENT

The authors would like to thank The University of Adelaide and Defence Science and Technology Group for their continued support.

REFERENCES

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc.*, 2014, pp. 338–342.

[4] E. A. Smirnov, D. M. Timoshenko, and S. N. Andrianov, "Comparison of regularization methods for imagenet classification with deep convolutional neural networks," *AASRI Proc.*, vol. 6, pp. 89–94, May 2014.

[5] T. Shaikhina and N. A. Khovanova, "Handling limited datasets with neural networks in medical applications: A small-data approach," *Artif. Intell. Med.*, vol. 75, pp. 51–63, Jan. 2017.

[6] E. Halilaj, A. Rajagopal, M. Fiterau, J. L. Hicks, T. J. Hastie, and S. L. Delp, "Machine learning in human movement biomechanics: Best practices, common pitfalls, and new opportunities," *J. Biomech.*, vol. 81, pp. 1–11, Nov. 2018.

[7] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[8] C. Kim, A. Misra, K. Chin, T. Hughes, A. Narayanan, T. Sainath, and M. Bacchiani, "Generation of large-scale simulated utterances in virtual rooms to train deep-neural networks for far-field speech recognition in Google home," in *Proc. INTERSPEECH*, 2017, pp. 379–383. [Online]. Available: http://www.isca-speech.org/archive/Interspeech_2017/pdfs/1510.PDF

[9] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2014, pp. 1–11.

[10] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Process. Lett.*, vol. 24, no. 3, pp. 279–283, Mar. 2017.

[11] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," in *Proc. ECML/PKDD Workshop Adv. Anal. Learn. Temporal Data*, Riva Del Garda, Italy, Sep. 2016.

[12] T. T. Um, F. M. J. Pfister, D. Pichler, S. Endo, M. Lang, S. Hirche, U. Fietzek, and D. Kulić, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," in *Proc. 19th ACM Int. Conf. Multimodal Interact. (ICMI)*. New York, NY, USA: ACM, 2017, pp. 216–220. doi: 10.1145/3136755.3136817.

[13] K. M. Rashid and J. Louis, "Times-series data augmentation and deep learning for construction equipment activity recognition," *Adv. Eng. Inform.*, vol. 42, Oct. 2019, Art. no. 100944. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474034619300886

[14] S. P. Moustakidis, J. B. Theocharis, and G. Giakas, "Subject recognition based on ground reaction force measurements of gait signals," *IEEE Trans. Syst., Man, Cybern., B (Cybern.)*, vol. 38, no. 6, pp. 1476–1485, Dec. 2008.

[15] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Anal. Chem.*, vol. 36, no. 8, pp. 1627–1639, 1964.

[16] E. Jones, T. Oliphant, and P. Peterson. (2001). *SciPy: Open Source Scientific Tools for Python*. [Online]. Available: http://www.scipy.org/

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[18] T.-H. Pham, S. Caron, and A. Kheddar, "Multicontact interaction force sensing from whole-body motion capture," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2343–2352, Jun. 2018.

JORDAN YEOMANS was born in Adelaide, Australia, in 1992. He received the bachelor's degree (Hons.) in mechanical and aerospace engineering from The University of Adelaide, Australia, in 2014. In 2018, he returned to The University of Adelaide, to complete his Master of Engineering with a focus on applying modern machine learning to the field. He is currently pursuing the Ph.D. degree in deep learning with Australian Institute for Machine Learning. His current research interests include reinforcement learning and decision abstraction.

SIMON THWAITES was born in Perth, WA, Australia, in 1991. He received the B.Eng. degree (Hons.) in mechanical and sports engineering and the M.Eng. degree in mechanical engineering from The University of Adelaide, SA, Australia, in 2013 and 2014, respectively, where he is currently pursuing the Ph.D. degree in medicine, under the supervision of Assoc. Prof. D. Thewlis. From 2015 to 2017, he was a Research Assistant with the School of Mechanical Engineering, and in 2018, he was a Research Officer with the School of Medicine, both from The University of Adelaide. His research interests include sports and clinical biomechanics, with a current focus on orthopaedics and trauma.

WILLIAM S. P. ROBERTSON was born in Adelaide, Australia, in 1981. He received the B.Eng. (Hons.) and Ph.D. degrees in mechanical engineering, in 2002 and 2013, respectively, under the supervision of B. Cazzolato, from The University of Adelaide, Australia, where he is currently a Lecturer with the School of Mechanical Engineering. His research interests include magnetic levitation, vibration control, and biomechanics. He has taught six courses in dynamics and control and sports engineering, and is a Lecturer and a Coordinator for the honours project course.

DAVID BOOTH received the B.Sc. degree in computer science from Wolverhampton University, in 1984, and the M.Phil. and Ph.D. degrees in applied mathematics from Oxford Brookes University, in 1986 and 1991, respectively. From 1986 to 2008, he had various roles at the Defence Science and Technology Laboratory, Malvern, U.K. and its predecessors. Since 2008, he has been a Senior Research Scientist with the Defence Science and Technology Group, Edinburgh, latter as part of the Biometrics Group. His current research interests include non-mainstream biometrics and video analytics.

BRIAN NG was born in Hong Kong, in 1974. He received the B.Sc. degree in mathematics and computer science, and the B.Eng. (Hons.) and Ph.D. degrees in electrical and electronic engineering, under the supervision of A. Bouzerdoum, from The University of Adelaide, Australia, in 1996, 1997, and 2003, respectively, where he is currently a Senior Lecturer with the School of Electrical and Electronic Engineering. His research interests include radar signal processing and wavelets, and terahertz (T-ray) signal processing. He is currently an active member of the South Australian Chapter of the IEEE. He received the University of Adelaide Medal for the Top Graduate in electrical and electronic engineering.

DOMINIC THEWLIS was born in Leeds, U.K. in 1982. He received the Ph.D. degree in biomechanics, in 2009, under the supervision of J. Richards from the University of Central Lancashire, U.K., where he was a Research Fellow from 2005 to 2009. He is currently an Associate Professor and National Health and Medical Research Fellow with Adelaide Medical School, The University of Adelaide. His research interests include biomechanics, computational modelling of the human musculoskeletal system, motion capture methods, pose estimation, biological signal processing, and machine learning. He is a member of the International Society of Biomechanics, and the Australian and New Zealand Orthopaedic Research Society.