SoftwareX 12 (2020) 100548

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

# Mcfly: Automated deep learning on time series

D. van Kuppevelt, C. Meijer, F. Huber, A. van der Ploeg, S. Georgievska, V.T. van Hees *

*Netherlands eScience Center, Science Park 140, 1098 XG, Amsterdam, The Netherlands*

ARTICLE INFO

*Article history:*
Received 24 January 2019
Received in revised form 13 April 2020
Accepted 12 June 2020

*Keywords:*
Deep learning
Machine learning
Time series
autoML
Research software

ABSTRACT

Deep learning is receiving increasing attention in the scientific community, but for researchers with no or limited machine learning experience it can be difficult to get started. In particular, the so-called hyperparameter selection, which is critical to successfully train a model, requires a good understanding of deep learning and some experience training models. We present *mcfly*, a Python package for deep learning for time series classification, designed to ease training by providing automated hyperparameter selection. We evaluated mcfly by organizing two workshops.



## Code metadata

| | |
|---|---|
| Current code version | v3.0.0 |
| Permanent link to code/repository used for this code version | https://github.com/ElsevierSoftwareX/SOFTX_2019_19 |
| Legal Code License | Apache-2.0 |
| Code versioning system used | git |
| Software code languages, tools, and services used | python, jupyter, javascript |
| Compilation requirements, operating environments & dependencies | Tested on Linux, Mac and Windows |
| If available Link to developer documentation/manual | https://github.com/NLeSC/mcfly/wiki/User-manual |
| Support email for questions | |

Current executable software version
Ancillary data table required for sub version of the executable software: (x.1, x.2 etc.) kindly replace examples in right column with the correct information about your executables, and leave the left column as it is.
Software metadata

| | |
|---|---|
| Current software version | v3.0.0 |
| Permanent link to executables of this version | https://github.com/NLeSC/mcfly/releases/tag/v3.0.0 |
| Legal Software License | Apache-2.0 |
| Computing platforms/Operating Systems | Linux, OS X, Microsoft Windows |
| Installation requirements & dependencies | python, pip |
| If available, link to user manual - if formally published include a reference to the publication in the reference list | https://mcfly.readthedocs.io/en/latest/ |
| Support email for questions | |

## 1. Motivation and significance

The recent rise of deep learning with many successful applications in classification and regression tasks came with a rapidly growing diversity of potential model architectures. Choosing most promising hyper-parameters and a well-performing model architecture has since become an art on its own that continues to be challenging, even for experts. This complicates taking the first steps towards applying deep learning for non-experts, such as quantitative scientists with little background in applying modern neural networks.

* Corresponding author.
*E-mail address:* v.vanhees@esciencecenter.nl (V.T. van Hees).

https://doi.org/10.1016/j.softx.2020.100548

These issues have led to an entirely new field of research on automated machine learning approaches (often termed autoML), aiming at (semi-) automating the search for well-performing machine-learning models or algorithms together with the automated choice of hyper-parameters [1]. Many current solutions aimed at non-experts are commercial software packages focused on business applications (e.g. Google autoML, Dataiku, Pienso).

Recently, a number of open source autoML software tools for scientists have appeared as well. Examples of such tools are Auto-WEKA [2], auto-sklearn [3], TPOT [4], H20 [5], or auto-Keras [6]. Those available tools give quick and simplified access to many standard machine learning algorithms (Auto-WEKA, auto-sklearn, TPOT, H2O), or image classification using deep learning (auto-Keras).

Mcfly[1] [7] is an autoML software tool that focuses on deep learning for multivariate times series classification. Time series classification is a challenge in various fields of research. The interest and potential of deep learning for time series is demonstrated by the growing number of publications, including: electroencephalography (EEG) signals [8,9]; audio signals [10]; electrocardiography (ECG) [11,12]; human activity recognition from movement sensor data [13,14], and machine health monitoring [15]. Recently it was shown that deep learning is able to outperform most classical classification techniques for both univariate and multivariate times series [16], on a wide range of problems. This comes at no surprise — the advantage of deep learning techniques is that they are able to work with raw data, compared to classical machine learning methods which take as input a condensed representation of the data engineered by a domain expert. Thus, deep learning methods have the potential to exploit more information from the input.

Mcfly was developed as an open source Python package that automates most of the decision making steps to make deep learning for time series accessible to a broader group of researchers. Mcfly is tailored towards people with no or little prior experience in applying machine learning and deep learning tools. This makes the package particularly suited for rapid model prototyping or educational purposes. However, due to its rich user interface, mcfly is also an easily configurable tool for researchers that are experienced in deep learning.

### 1.1. Related work on tools for deep learning for time series

Recently, a few other tools for classification of time series have been released, notably [17] and [18]. The former by Löning et al. implements various algorithms for time series classification, including the deep learning architectures treated in the overview [16]. In [18] Fawaz et al. propose a deep learning architecture that is based on the Inception neural network used in computer vision and show that it is competitive with state-of-art non-deep learning methods, while providing significantly faster training, on a wide range of time series classification problems.

Contrary to [18] or [16], mcfly provides the user the ability to choose particular ranges of values for the hyper-parameters (depending on domain knowledge and data size) or to explore a large number of networks generated with various hyper-parameters, in case of limited domain knowledge. In the latter case the user can use the best performing network as a starting point for more fine-tuning and training. We emphasize, however, that domain knowledge is critical in architecting the best performing machine learning algorithm,[2] as a priori all algorithms are equivalent [19]. This is why the main purpose of mcfly is to provide an easy start to domain experts new to deep learning and an easily configurable tool to researchers that are more advanced in deep learning.

Furthermore, mcfly has the advantage that, from a developer's perspective, the tool is developed as a software package following professional software standards [20,21]. The code base has been covered 88% by unit tests, supporting continuous integration and easy addition of new features and neural network architectures. This flexibility increases the maintainability of mcfly, as well as the reproducibility and transparency of research results produced with mcfly.

## 2. Software description

Mcfly is a wrapper around the Keras API in Tensorflow, a deep learning framework in Python [22]. Although the Keras API in Tensorflow is a powerful and user-friendly API, it does require the user to define the architecture of the model and other hyperparameters, e.g. learning rate. Mcfly eases this step by proposing an architecture and other hyperparameters based on a random search over a range of plausible settings. (It has been shown in [23] that random-search is more efficient than the other commonly used technique, grid-search.) Since the resulting models are Keras models, it is possible to re-use and advance the architecture and trained model with the full functionality of the Keras API in Tensorflow. In addition, utilities like saving and loading models can be directly used from the Keras API.

The workflow for users of mcfly typically consists of the following stages: (1) prepare input data, (2) generate model(s), (3) find best model, and (4) visualize the process. The most commonly used functions in this workflow are available from the top level of the python package. The implementation of all functions is divided into the modules `modelgen` and `find_architecture`. In addition, we developed a Jupyter notebook tutorial that demonstrates how mcfly is used.

### 2.1. Preparing input data

Before using the mcfly functionalities, the user should prepare their data so that mcfly accepts it as input.

The input data format accepted by mcfly is identical to the input data format accepted by the Keras API, but limited to matrices representing single or multi-channel time series data. Each (real valued) multi-channel time series sequence is associated with one class label. All sequences in the dataset should be of equal length. Further, the input data is expected to be split into a training, test, and validation set. For each of the sets, the input X and the output y are both numpy arrays. The input data X should be of shape (num_samples, num_timesteps, num_channels). The output data y is of shape (num_samples, num_classes), as a binary array for each sample. We recommend storing the numpy arrays as binary files with the numpy function np.save.

If the original time series has multiple labels per time window the user is expected to use a sliding window and to provide a representative label for each window. Since the choices necessary for window size selection and other preprocessing steps are very application dependent, mcfly does not provide functions for this stage, although a few recommendations are given in the documentation.

[1] https://github.com/NLeSC/mcfly.

[2] For example, time series consisting of short patterns interspersed with long random noise would benefit from convolutional networks with small filters in the first layer, while time series consisting of long patterns without random noise would benefit from longer filters.

### 2.2. Model generation

The next step is to generate a list of untrained Keras models based on the shape of the input data, which is done with function *generate_models*. Key function arguments include *number_of_models* to specify the number of models that need to be generated, *x_shape* to specify the shape of the input training data, and *number_of_classes* to specify the number of classes in the data labels. An additional set of arguments, with default values, provide more control for advanced users.

There are four types of network architectures that are available in mcfly: CNN, DeepConvLSTM, ResNet and InceptionTime. The details of these architectures are discussed in the next subsections. The argument *model_types* gives the user the opportunity to specify which architectures to choose from, and by default all four model types are used. The model generation function chooses the model type at random, but in an alternating order so that the number of models per model type is roughly equal. The first layer in all architectures is a Batchnorm layer, so that the user does not have to normalize the input data.

For a number of model choices we follow common practice: Weight initialization is done with LeCun Uniform [24], kernel size is 3, all convolutional and dense layers use L2 regularization [25], categorical cross-entropy is used as loss function [26], accuracy is by default outputted and used as a performance metric to choose the best performing model. Note that the performance metric can be changed with argument *metric* in most mcfly functions. The default, but modifiable, log range for the learning rate and the regularization rate is $[10^{-1}, 10^{-4}]$.

#### 2.2.1. Convolutional Neural Network

The model type CNN is a regular Convolutional Neural Network, with N convolutional layers with ReLU activation [27] and one hidden dense layer. An example of the application of CNNs on time series data is in [28]. The network architecture generated by mcfly is illustrated in Fig. 1 (top).

The hyperparameters of the CNN are the number of Convolutional layers (default range to choice from: [1, 10]), as well as the number of filters (default range: [10, 100]) in each Convolutional layer and the number of neurons in the hidden Dense layer (default range [10, 2000]). We decided not to add Pool layers because reducing the spatial size of the sequence is usually not necessary if there are enough convolutional layers.

#### 2.2.2. Deep convolutional LSTM

The architecture of the model type DeepConvLSTM is based on the paper by Ordóñez et al. [29], originally developed for multimodal wearable activity recognition. The model combines Convolutional layers with Long short-term memory (LTSM) layers. The generated LSTM network architecture is illustrated in Fig. 1 (bottom). In contrast to the CNN model, the convolutional layers in the DeepConvLSTM model are applied per channel, and connected in the first LSTM layer. Applying convolutions per channel was reported to be suitable for activity recognition tasks based on wearable sensor data where the channels are not necessarily synchronized in time (each channel corresponds to a different body part) [29]. The LSTM layers result in a multidimensional vector per time step. The Timedistributed Dense layer is used to output a sequence of predictions, and the TakeLast layer to pick the last element from the sequence as the final prediction. The hyperparameters of the DeepConvLSTM architecture are the number of convolutional layers (default range to choose from: [1,10]), the number of LSTM layers (default range: [1,5]), the number of filters for each Conv layer (default range: [10, 100]) and the hidden layer dimension for each LSTM layer (default range: [10, 100]).

#### 2.2.3. ResNet

Many convolutional architectures are inspired by the field of image classification where deep learning had its first and biggest breakthroughs. One very successful technique for dealing efficiently with deeper networks are so called skip-connections (also termed residual connections) which are connections that skip multiple layers within deeper neural networks [30]. This neural network architecture has also been adapted for time series classification [31] and was shown to perform comparably well in a large number of cases [16]. One of the hyperparameters of the ResNet architecture are the number of residual modules (default range to choose from: [2, 5]). One such module consists of three convolutional layers and one skip connection bridging all three layers. Two other key hyperparameters are the number of filters and the kernel size for the convolutions. Following the original ResNet design for time series classification [31], we chose a maximum kernel size (default range: [8, 32]) and derive the kernel sizes for the different levels by scaling down by $2^{-i/2}$ for i the index of the residual module. Analogously, a minimum number of filters is chosen (default range: [32, 128]) based on which the numbers of filters for all residual modules is derived following $2^{i/2}$ with i the index of the residual module.

#### 2.2.4. InceptionTime

Another architecture element that turned out to be very helpful in handling neural networks of greater depth and width is inception modules [32]. Inception modules run convolutions with different kernel sizes in parallel and then combine the outcome. While initially applied to image classification problems, their adaptation for time series classification was recently seen to deliver very promising results on a wide variety of datasets [18].

As key hyper-parameters based on the InceptionTime architecture we picked the number of Inception modules (default range to choose from: [3, 6]). Each such module consists of a bottleneck layer (max pooling) followed by three convolutional layers with varying kernel sizes and one convolution with kernel size $= 1$. Mcfly randomly chooses a maximum kernel size (default range: [10, 100]) based on which the kernel sizes for the different layers are derived by scaling down by dividing by 2 and 4. Another key parameters is the number of filters which is the same for all layers in is chosen (default range: [32, 96]).

### 2.3. Finding the best model

The model architectures generated in the previous step are compared with the function *train_models_on_samples* by training them on the training data (or possibly a subset thereof) and evaluating the models on the validation subset. By looking for the best performing model mcfly effectively performs a random search over the hyper parameter space. We chose to implement random search, because it is simple and fairly effective [23]. This will help us to choose the best candidate model. The performance results for the models are stored in a json file, which we will visually inspect later on. There are a few optional settings for training the models in mcfly, such as the number of epochs and whether the training process should stop when performance does not increase (*early_stopping*).

### 2.4. Visualization

In addition to the python modules, we developed a web-based visualization of the training process. The function *train_models_on_samples* optionally writes the model architectures and the performance of the modules during the training process to a json file. This file can be uploaded in the visualization tool, which then provides an interactive comparison of the different models. The

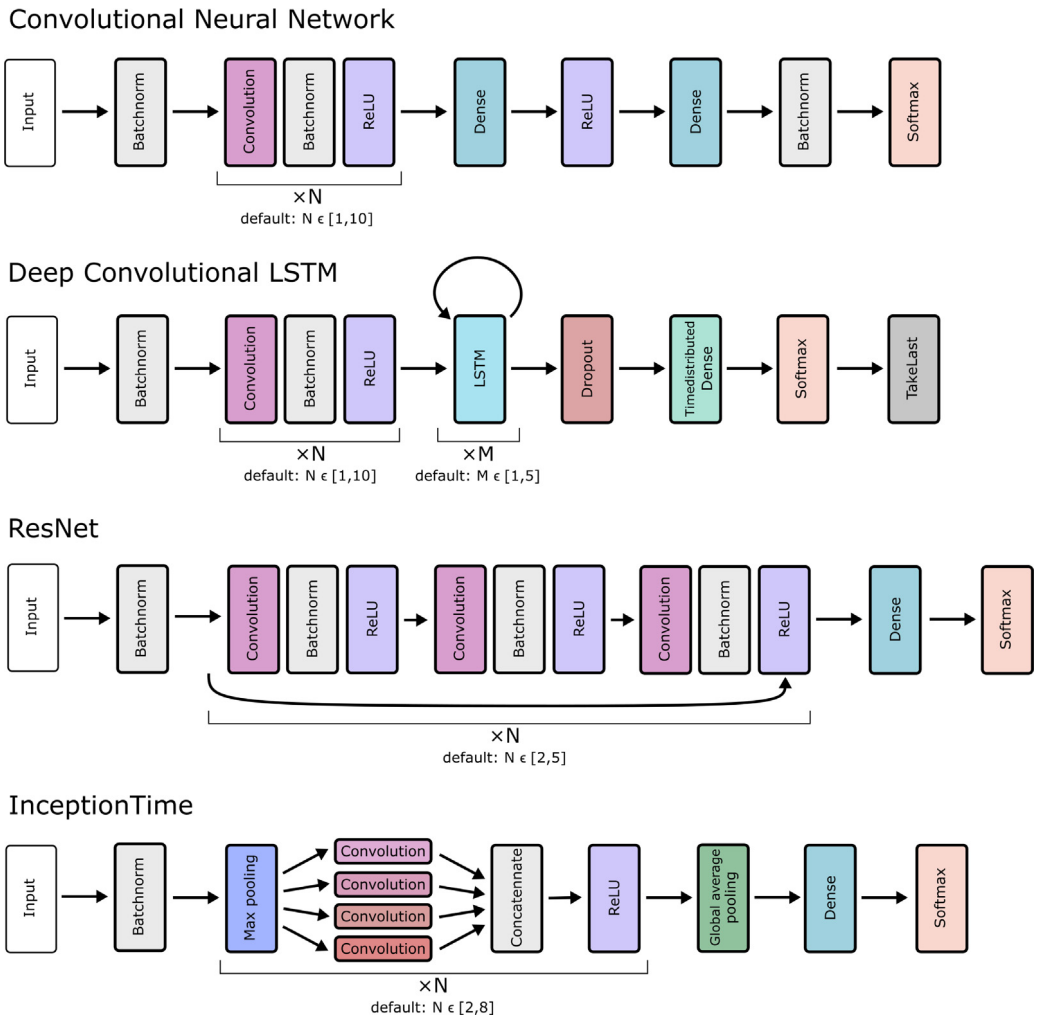


**Fig. 1.** Architectures of the convolutional, LSTM, ResNet, and InceptionTime networks generated by mcfly.

visualization is built in JavaScript, using the libraries dc and d3 for interactive plotting. The visualization is hosted through github pages.[3]

### 2.5. Tutorial

To strengthen the educational value of mcfly, we developed a tutorial.[4] The tutorial consists of a Jupyter notebook with explanations on mcfly and step-by-step code blocks to apply it on a real-life dataset.

### 2.6. Sample code snippets

The following code generates ten models that can be applied to the training data:

```
models = mcfly.generate_models(x_shape = X_train.shape,
                number_of_classes=y_train_binary.shape[1],
                                    number_of_models = 10)
```

The following code evaluates the models that were generated in the previous step:

```
histories, val_accuracies, val_losses =
      mcfly.train_models_on_samples(
          X_train, y_train_binary, X_val, y_val_binary,
           models,nr_epochs=5,
           subset_size=300,
           verbose=True,
           outputfile=outputfile)
```

## 3. Illustrative examples

The example we show in this section is based on the same data as used in the mcfly tutorial. We use the Physical Activity Monitoring (PAMAP2) dataset [33,34]. In short, this dataset consists of recordings of 3 acceleration sensors at 3 body locations (wrist, chest and ankle), from 9 participants that performed, in total, 12 activity types, such as lying, sitting and cycling. The data is preprocessed in a similar manner as in the original paper that released the data [33,34]. Time segments are constructed by applying a sliding window of 5.12 s on the 100 Hz data. Only the 7 activities that were performed by all subjects are selected. Time segments that are near to the border between activities are disregarded. This results in a total of 11597 data points, where each data point is a segment of length 512 and depth 9.

As described in Section 2.1, the data input for mcfly are numpy arrays for training, validation and test data. We leave out 100 segments for validation and 100 segments for training. This way of dividing the data was chosen for illustration purposes.

[3] http://nlesc.github.io/mcfly/.
[4] https://github.com/NLeSC/mcfly-tutorial

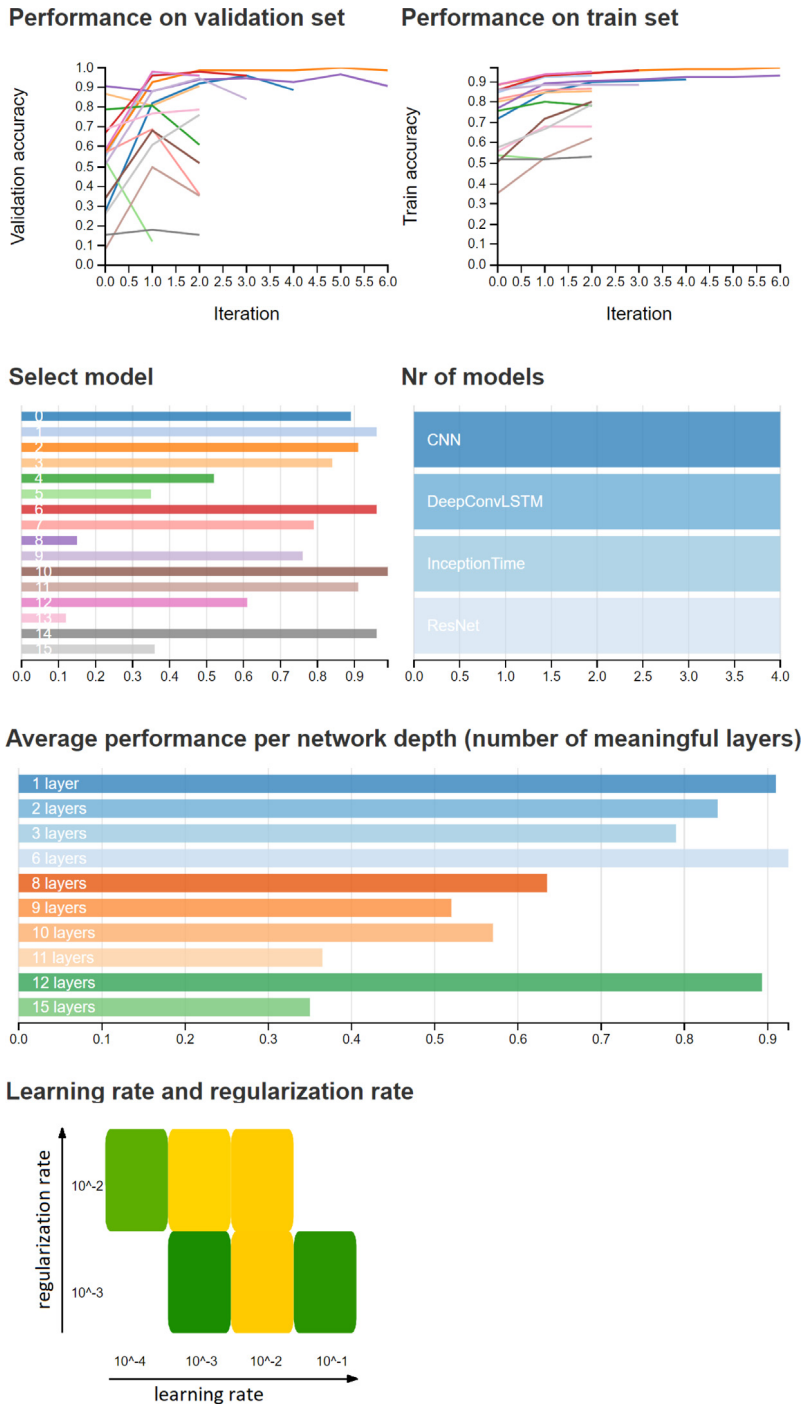


**Fig. 2.** Screenshot of the interactive visualization. See http://nlesc.github.io/mcfly/ to get a better impression of the full functionality..

To speed up the tutorial we stored the pre-processed data online [35], and we created a function to load this data:

```
X_train, y_train_binary, \
 X_val, y_val_binary, \
 X_test, y_test_binary, labels = \
  tutorial_pamap2.load_data(data_path)
```

Next, the tutorial uses the *modelgen.generate_models* to generate a series of models. We show an example of 15 models that are trained with early stopping turned on. The result of the training process is shown in the interactive visualization, see Fig. 2. The visualization shows a wide range in performance between the models.

## 4. Impact

Mcfly is, to the best of our knowledge, the only open source tool for deep learning on time series data aimed at novices in machine learning. Its main value is to provide an environment to quickly apply deep learning classification of time series data, relieving the user from the technicalities of deep neural networks architectures and training.

To increase the visibility of mcfly, we organized two workshops for academic researchers to teach the fundamentals of neural networks, together with a hands-on tutorial of mcfly based on the notebook tutorial. At the end of each 3-hour workshops, all participants were able to train a neural network on the PAMAP2

dataset. In 2 months after the workshop we sent out a short survey to all 30 participants. Out of the 14 participants that filled in our survey afterwards, 12 considered their knowledge about deep learning prior to the workshop 'beginner level'. Response to other questions are: "The workshop was useful for my research/work" (10 yes, 3 neutral, 1 no), "I am happy with the online training materials" (12 yes, 2 neutral, 0 no), and "the workshop improved my skills on the topic" (11 yes, 2 neutral, 1 no).

The dissemination of mcfly has further been promoted with a blog post[5] and a presentation at the PyData conference in Amsterdam, available on YouTube.[6] Further, the web page: https://www.research-software.nl/software/mcfly was created to increase mcfly's findability and accessibility. All activities together resulted in a significant user base, over 270 stars and over 75 forks of the Github repository, and attention from the online community.[7,8] We expect that the user base grows further as we continue our efforts to disseminate and upgrade the software.

So far, mcfly has been used in internet of things, to improve pay-per-use pricing in the context of servitization for industrial manufacturing firms [36] and in physics, for classification of diffusion modes in single-particle tracking data [37]. In addition, it is being used in research in progress [38] in genomics, for detecting somatic structural variants in the human cancer genome, thus proving effectiveness on other sequential data.

## 5. Conclusions

We introduced mcfly, a python package for deep learning on time series. Mcfly lowers the barrier for researchers new to deep learning, by automating the process of architecture and hyperparameter selection. Within the growing field of autoML tools, mcfly is unique in that it focuses on time series classification. It is particularly suited for rapid model prototyping and educational purposes, but is also configurable for advanced users.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

The project was paid for by the Netherlands eScience Center.

## References

[1] Guyon I, Sun-Hosoya L, Boullé M, Escalante HJ, Escalera S, Liu Z, et al. Analysis of the AutoML challenge series 2015-2018. In: Hutter F, Kotthoff L, Vanschore J, editors. AutoML: Methods, systems, challenges. The springer series on challenges in machine learning, Springer Verlag; 2018, p. 88, URL https://hal.archives-ouvertes.fr/hal-01906197.

[2] Kotthoff L, Thornton C, Hoos HH, Hutter F, Leyton-Brown K. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. J Mach Learn Res 2017;18(1):826–30, URL http://dl.acm.org/citation.cfm?id=3122009.3122034.

[3] Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F. Efficient and robust automated machine learning. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R, editors. Advances in neural information processing systems 28. Curran Associates, Inc.; 2015, p. 2962–70, URL http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.

[4] Olson RS, Urbanowicz RJ, Andrews PC, Lavender NA, Kidd LC, Moore JH. Automating biomedical data science through tree-based pipeline optimization. In: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I. Springer International Publishing; 2016, p. 123–37. http://dx.doi.org/10.1007/978-3-319-31204-0_9.

[5] The H2Oai team. h2o: Python interface for H2O. 2015, Python package version 3.1.0.99999. URL http://www.h2o.ai.

[6] Jin H, Song Q, Hu X. Auto-keras: Efficient neural architecture search with network morphism. 2018, arXiv:cs.LG/1806.10282.

[7] van Kuppevelt D, Meijer C, van Hees V, Bos P, Spaaks J, Kuzak M, et al. mcfly: deep learning for time series. Zenodo; 2019, http://dx.doi.org/10.5281/zenodo.2541698.

[8] Ahmedt-Aristizabal D, Fookes C, Nguyen K, Sridharan S. Deep classification of epileptic signals. In: Conference proceedings : ... Annual international conference of the IEEE engineering in medicine and biology society. IEEE engineering in medicine and biology society. Annual conference, vol. 2018, 2018, p. 332–5. http://dx.doi.org/10.1109/EMBC.2018.8512249, URL http://www.ncbi.nlm.nih.gov/pubmed/30440405.

[9] Kim S, Kim J, Chun H-W. Wave2Vec: Vectorizing electroencephalography bio-signal for prediction of brain disease. Int J Environ Res Public Health 2018;15(8). http://dx.doi.org/10.3390/ijerph15081750, URL http://www.ncbi.nlm.nih.gov/pubmed/30111710. http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6121271.

[10] Kim Y, Sa J, Chung Y, Park D, Lee S. Resource-efficient pet dog sound events classification using LSTM-FCN based on time-series data. Sensors 2018;18(11). http://dx.doi.org/10.3390/s18114019, URL http://www.ncbi.nlm.nih.gov/pubmed/30453674.

[11] Liu M, Kim Y. Classification of heart diseases based on ECG signals using long short-term memory. In: Conference proceedings : ... Annual international conference of the IEEE engineering in medicine and biology society. IEEE engineering in medicine and biology society. Annual conference, vol. 2018, 2018, p. 2707–10. http://dx.doi.org/10.1109/EMBC.2018.8512761, URL http://www.ncbi.nlm.nih.gov/pubmed/30440962.

[12] Yao Z, Chen Y. Arrhythmia classification from single lead ECG by multi-scale convolutional neural networks. In: Conference proceedings : ... Annual international conference of the IEEE engineering in medicine and biology society. IEEE engineering in medicine and biology society. Annual conference, vol. 2018, 2018, p. 344–7. http://dx.doi.org/10.1109/EMBC.2018.8512260, URL http://www.ncbi.nlm.nih.gov/pubmed/30440407.

[13] Zebin T, Sperrin M, Peek N, Casson AJ. Human activity recognition from inertial sensor time-series using batch normalized deep LSTM recurrent networks. In: Conference proceedings : ... Annual international conference of the IEEE engineering in medicine and biology society. IEEE engineering in medicine and biology society. Annual conference, vol. 2018, 2018, p. 1–4. http://dx.doi.org/10.1109/EMBC.2018.8513115, URL http://www.ncbi.nlm.nih.gov/pubmed/30440301.

[14] Kang J, Lee J, Eom D-S. Smartphone-based traveled distance estimation using individual walking patterns for indoor localization. Sensors 2018;18(9). http://dx.doi.org/10.3390/s18093149, URL http://www.ncbi.nlm.nih.gov/pubmed/30231534. http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6165575.

[15] Qiao H, Wang T, Wang P, Qiao S, Zhang L. A time-distributed spatiotemporal feature learning method for machine health monitoring with multi-sensor time series. Sensors 2018;18(9). http://dx.doi.org/10.3390/s18092932, URL http://www.ncbi.nlm.nih.gov/pubmed/30177670. http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6164508.

[16] Ismail Fawaz H, Forestier G, Weber J, Idoumghar L, Muller P-A. Deep learning for time series classification: a review. Data Min Knowl Discov 2019;33(4):917–63. http://dx.doi.org/10.1007/s10618-019-00619-1, URL https://doi.org/10.1007/s10618-019-00619-1.

[17] Löning M, Bagnall AJ, Ganesh S, Kazakov V, Lines J, Király FJ. Sktime: A unified interface for machine learning with time series. 2019, ArXivabs/1909.07872.

[18] Fawaz HI, Lucas B, Forestier G, Pelletier C, Schmidt DF, Weber J, et al. InceptionTime: Finding alexnet for time series classification. 2019, ArXivabs/1909.04939.

[19] Wolpert DH. The lack of a priori distinctions between learning algorithms. Neural Comput 1996;8(7):1341–90. http://dx.doi.org/10.1162/neco.1996.8.7.1341.

[20] Maassen J, Drost N, Van hage W, van Nieuwpoort R. Track 2 lightning talk: Software development best practices at the Netherlands eScience center. figshare; 2017, http://dx.doi.org/10.6084/m9.figshare.5327587.v2, URL https://figshare.com/articles/nlesc_pdf/5327587/2.

[21] Drost N, Spaaks JH, Andela B, Veen L, van der Zwaan JM, Verhoeven S, et al. Software development guide. URL https://github.com/NLeSC/guide.

[22] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015, Software available from tensorflow.org. URL https://www.tensorflow.org/.

[23] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. J Mach Learn Res 2012;13(Feb):281–305.

[5] https://blog.esciencecenter.nl/mcfly-time-series-classification-made-easy-e47de8d29838

[6] https://www.youtube.com/watch?v=9X_4i7zdSY8

[7] https://twitter.com/BertrandDechoux/status/909874216480182272

[8] https://stackoverflow.com/questions/45017312

[24] LeCun Y, Bottou L, Orr GB, Müller K-R. Efficient backProp. In: Neural networks: Tricks of the trade. Springer; 1998, p. 9–50. http://dx.doi.org/10.1007/3-540-49430-8_2, URL http://link.springer.com/10.1007/3-540-49430-8_2.

[25] Ng AY. Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In: Twenty-first international conference on machine learning. New York, New York, USA: ACM Press; 2004, p. 78. http://dx.doi.org/10.1145/1015330.1015435, URL http://portal.acm.org/citation.cfm?doid=1015330.1015435.

[26] Mannor S, Peleg D, Rubinstein R. The cross entropy method for classification. In: Proceedings of the 22nd international conference on machine learning. New York, New York, USA: ACM Press; 2005, p. 561–8. http://dx.doi.org/10.1145/1102351.1102422, URL http://portal.acm.org/citation.cfm?doid=1102351.1102422.

[27] Nair V, Hinton GE. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning. 2010. p. 807–14.

[28] Martinez HP, Bengio Y, Yannakakis GN. Learning deep physiological models of affect. IEEE Comput Intell Mag 2013;8(2):20–33.

[29] Ordóñez F, Roggen D. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. Sensors 2016;16(1):115. http://dx.doi.org/10.3390/s16010115, URL http://www.mdpi.com/1424-8220/16/1/115.

[30] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition. 2015. p. 770–778.

[31] Wang Z, Yan W, Oates T. Time series classification from scratch with deep neural networks: A strong baseline. In: 2017 international joint conference on neural networks. 2016. p. 1578–1585.

[32] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. 2014, arXiv:1409.4842.

[33] Reiss A, Stricker D. Creating and benchmarking a new dataset for physical activity monitoring. In: Proceedings of the 5th international conference on pervasive technologies related to assistive environments. New York, New York, USA: ACM Press; 2012, p. 1. http://dx.doi.org/10.1145/2413097.2413148, URL http://dl.acm.org/citation.cfm?doid=2413097.2413148.

[34] Reiss A, Stricker D. Introducing a new benchmarked dataset for activity monitoring. In: 2012 16th international symposium on wearable computers. IEEE; 2012, p. 108–9. http://dx.doi.org/10.1109/ISWC.2012.13, URL http://ieeexplore.ieee.org/document/6246152/.

[35] van Kuppevelt D, van Hees V, Meijer C. Pamap2 dataset preprocessed v0.3.0. 2017, http://dx.doi.org/10.5281/zenodo.834467.

[36] Heinis TB, Loy CL, Meboldt M. Improving usage metrics for pay-per-use pricing with IoT technology and machine learning. Res-Technol Manag 2018;61(5):32–40. http://dx.doi.org/10.1080/08956308.2018.1495964, URL https://www.tandfonline.com/doi/full/10.1080/08956308.2018.1495964.

[37] Kowalek P, Loch-Olszewska H, Szwabiński J. Classification of diffusion modes in single-particle tracking data: Feature-based versus deep-learning approach. Phys Rev E 2019;100:032410. http://dx.doi.org/10.1103/PhysRevE.100.032410, URL https://link.aps.org/doi/10.1103/PhysRevE.100.032410.

[38] Santuari L, Georgievska S, Shneider C, Kuzniar A, Schaefers T, Klooster-man W, et al. DeepSV: Somatic structural variant detection with deep learning. 2018, http://dx.doi.org/10.5281/zenodo.1254881.