Supplementary Material for
Uyulan, Ç, Mayor, D., Steffert, T., Watson, T., Banks, D. Classification of the central effects of transcutaneous electro-acupuncture stimulation (TEAS) at different frequencies: A Deep Learning approach using wavelet packet decomposition with an entropy estimator. Applied Sciences **2023**, *13*, 2703.https://doi.org/10.3390/app13042703.

Online Supplementary Material

[**SM1**] **A brief overview of AI: machine learning (ML) and deep learning (DL) in EEG analysis**

[**SM1.1**] *A rough sketch of ML methods used for EEG data analysis*

We live today in a world of big data. Finding patterns in such data to classify the data or predict future outcomes is increasingly the task of artificial intelligence (AI), in particular machine learning (ML). In contrast to classical methods of analysis such as statistics, where knowledge methods are defined in a relatively fixed way [57], ML computer algorithms are created that can learn from their mistakes and improve through experience. Such learning may be 'supervised', 'unsupervised' or 'semi-supervised'. In supervised learning, the algorithm is 'trained' on a set of previously labelled input data to create a 'model' or template for learning that will then be able to estimate outputs for unseen, unlabelled data in the future. In unsupervised learning, the algorithm is used to uncover patterns (e.g., trends, subgroups or outliers) in unlabelled or unclassified input data. 'With unsupervised learning, there is no right or wrong answer' [58]. Semi-supervised learning combines the two, but with more unlabelled than labelled data. Whichever approach is taken, informative input features are identified in a process termed 'feature selection' (either manually, even 'hand-crafted', based on expert-level knowledge or trial-and-error [59,60], or by the algorithm itself, without requiring domain expertise), and then analysed using a mapping function that generates output predictions from these features [61].

ML algorithms used for classification ('classifiers') include (1) 'base' classifiers, (2) 'meta-methods' using meta-features from the data that enable a data-driven rather than hand-crafted approach [62], and (3) ensemble methods that take as input several 'base' classifiers and their parameters [63]. Some methods, such as the popular AdaBoost, can be considered as ensemble meta-methods, so combining the attributes of two families of a classifier. From the papers located in PubMed on ML and EEG, the most frequently used supervised ML algorithm (base classifier) appears to be the Support Vector Machine (SVM) [64] (289 hits), followed by the ensemble Random Forest (RF) method [65] (98 hits), with base classifiers Linear discriminant analysis (LDA) [66] (78 hits) and Logistic regression (LR, not to be confused with Linear Regression) [67] (56 hits) being the next most popular. Several review papers confirm that these are the most commonly used approaches in ML (e.g., [68]). SVM in its simplest form is a binary linear classifier but has many variants [69]. RF is a popular and powerful method, 'ensemble' because it is based on multiple decision trees, using Bootstrap Aggregation (or 'bagging') [70], where data is repeatedly resampled, and a final consensus decision taken. LDA is useful where groups are predetermined [71]. LR is often considered inferior to SVM and may require regularisation for optimum results [72].

Unsupervised ML algorithms include clustering (98 hits) (e.g., microstate analysis [73]) and principal component analysis (PCA) (46 hits)] [74]. A useful overview of algorithms is provided by Jason Brownlee [75].

Hybrid models may combine several different ML methods, both supervised and unsupervised [76].

A common issue in ML, and more particularly in DL, is 'overfitting' when a model shows more accuracy on the training data but less on the test data or unseen data [77]. This may occur, for example,

when the model is too complicated and as a result learns more about the particular details (e.g., noise, random error) of the training data than the genuine relationships that may exist within the data. The model then becomes incapable of generalising to other input. If more data is not available for training, it may be necessary to reduce the complexity of the model, remove unnecessary features, use 'cross-validation' (splitting the training data into multiple smaller 'folds') or 'regularisation' (shrinking some model parameters, even to zero, using a separate tuning parameter to 'penalise' the flexibility of the model) [78].

A typical ML (or DL) pipeline consists of four main phases or steps, as depicted in Figure S1**.** After data are 'ingested', they may be transformed in some way, for instance, normalised or standardised, as in any other computation pipeline, to best expose their unknown underlying structure to the learning algorithms to be used [79]. The data (or the extracted features) will also need to be validated, to check that the pipeline is appropriate for their analysis. To improve results, 'Features' (i.e., particular characteristics) may also be extracted from the data – in a process called 'feature extraction', 'feature engineering' or 'featurisation' – rather than simply inputting the raw data.

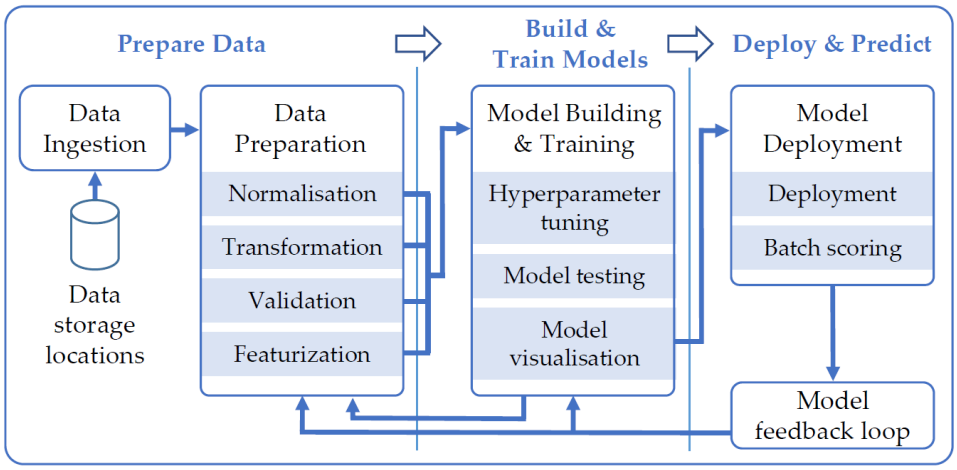

Figure S1. The main phases of a typical ML/DL pipeline: Data 'ingestion' and preparation, Model building and training, and Model deployment and prediction, with scoring assessment and feedback loops (Tony Steffert, adapted from Lazzeri 2019 [80]).

Also important will be 'Feature selection' (selecting the best subset of features to use), or other methods of 'dimensionality reduction', required in to speed up training and avoid the so-called 'curse of dimensionality' – the result of trying to analyse small amounts of data with many variables. The volume of the 'feature space' then ends up being very large, and the data in that space are sparsely scattered, thus representing only a small and very likely non-representative sample [81,82].

Once the data is prepared, the next step is to choose a model and then train it. The model can be selected based on prior knowledge or using an automated method. The model's 'hyperparameters' will also need tuning – default hyperparameters are highly unlikely to work well initially, as are the model parameters, estimated or learned from the data and often saved as part of the model's specific internal settings [83]. Algorithm hyperparameters and some well-known automated methods of selecting models and tuning their hyperparameters are mentioned in Sections SM1.3 and SM1.6, below.)

If the dataset is sufficiently large, a training set can be randomly selected from it, and the remaining data – with the same probability distribution as the training set – 'held out' unseen for testing the model once it is developed (usually an iterative process). This is to evaluate the performance of the model on data that was not used to train the model. Using the train-test split approach may also help with computational efficiency. There is no generally fixed train-to-test proportion [84], but a 2/3 to 1/3 split

can be a good place to start [85]. A further dataset, also unseen and sometimes called the validation dataset, can be used to check that 'data leakage' has not occurred (in other words, that information from outside the training dataset has somehow been used to create the model) [84], and to tune the model hyperparameters. Confusingly, sometimes the term 'validation set' is used to refer to what is described here as the 'test set' [86]. Figure S2 shows the overall training, validation, and test set pipeline.

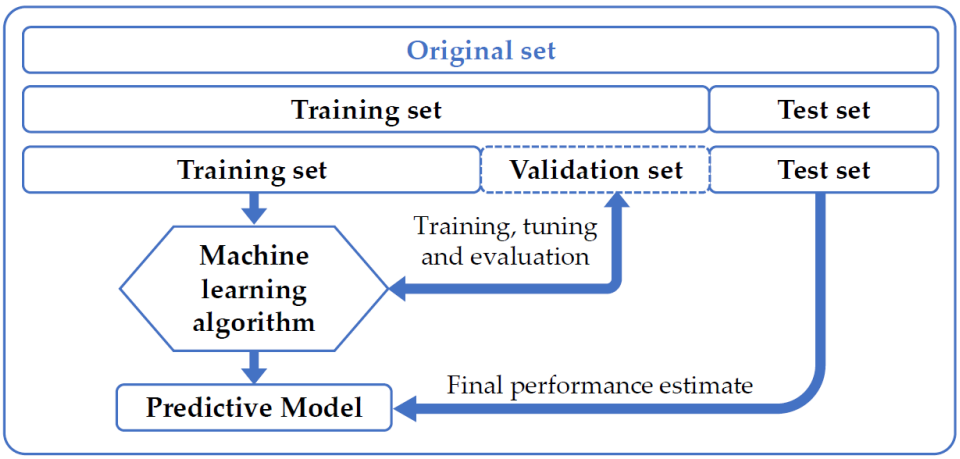

Figure S2. Training, validation, and test set pipeline. The percentages shown are those used in this study. (Tony Steffert, adapted from Ng 2020 [87]).

For smaller datasets, the resampling method of *k*-fold cross-validation is generally used to estimate how well the model performs, instead of a train-test split. This involves randomly dividing the set of observations into *k* groups, or 'folds,' of approximately equal size, the first fold being treated as a validation set, and the method being fitted on the remaining ($k - 1$) folds. Values of *k* are often selected as 5 or 10, as these have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance [84].

Once the model has been trained, it must be evaluated. One relatively straightforward way of doing this is to use a 'confusion matrix' (see below, p. 5).

Some advantages and disadvantages of the main ML algorithms used in EEG research are listed in **Table S1**.

Table S1. Some advantages and disadvantages of the main ML algorithms used in EEG research. (See Abbreviations list in paper for interpretation.)

Algorithm	Some advantages	Some disadvantages
SVM	Linear and stable [88]. Widely considered the most powerful training method. Generalises well, useful for small data [89], but also robust for multi-column data [90,91]. Low computational complexity [35]. Robust against overtraining [92] and outliers [90]; binary [64] but may be adapted for multiclass classification [93]. Less prone to overfitting than some other methods [94]. Although primarily for classification, SVM may also be adapted for regression analysis [95].	Poor learning for nonlinear data (but can use with nonlinear kernel function), performs poorly for abnormal or noisy data [91]; multiclass classification have usually to be reduced to multiple binary classifications. Slow for large data (high algorithmic complexity and extensive memory requirements [94]); does not perform well if classes overlap; selecting appropriate kernel function and hyperparameters is important and may not be easy [90]; a 'black box' whose

		inner workings may be hard to follow [58].
DT	Simple to understand, visualise and interpret; able to handle both numerical and categorical data; performs well even if assumptions are somewhat violated by the true model from which data were generated [70]; computationally cheap, able to train data that contains errors or missing values [96; construction does not require specialist domain knowledge; can handle high dimensional data [94].	Risk of overfitting; may be unstable; may not return the *globally* optimal decision tree [need to use within an ensemble learner]; tree biased if some classes dominate need to balance dataset] [70]; requires a large volume of data for accurate results [96]; output attribute must be categorical and limited to one output attribute [94].
RF	Can handle highly correlated features, outliers, and missing data; performs well on imbalanced data; no problem with overfitting [90] compared with DT (Decision Tree); easier to understand than SVM; useful for small training samples [97] or large datasets [65]; can handle both discrete and continuous data [35]; can be used for classification or regression; provides estimates of which variables are important for classification [65].	'Black box,' so hard to understand process [90].
LDA	Linear and stable [88]; supports both binary and multi-class classification [98]; may be used with small samples [99].	Requires linear, normally distributed data; not robust to outliers or noise [100]; the size of the smallest group must be larger than the number of predictor variables [71].
LR	Simple, effective, input features do not need rescaling, hyperparameters do not require tuning [90]; computationally efficient, ease of regularisation; robust to noise and multicollinearity [91].	Not powerful; does not perform well if features highly correlated or irrelevant [90]; unstable if classes well separated; for binary classification only [101], but multinomial forms also exist [102]; prone to overfitting, requires all independent variables to be identified [91]; cannot be used with small samples [103].
Clustering	*k*-means clustering is the most popular clustering method [104], easy to implement, suited to large datasets [105]; ease in implementation, interpretation; computationally efficient [91].	Hierarchical clustering is inflexible and computationally costly; *k*-means is 'greedy,' depending on initial and local conditions [105], so may not find the best solution; prediction of k is hard (no unique solution for a certain k); sensitive to outliers [91].
k-NN	*k*-Nearest Neighbours is simple, nonlinear, and stable; computational complexity decreases with increasing k; lends itself very easily to parallel implementations [94]; flexible [91].	Sensitive to the local structure of the data, so degraded by noisy or irrelevant features; does not cope with data of large dimension ('highly susceptible to the curse of dimensionality' [94]);

		increasing *k* decreases performance; large storage requirements [94].
NB	Naïve Bayes nonlinear, stable; simple, low computation cost; can cope with large dimension or multiple classes; training and classification can be accomplished with a single pass over the data; robust to noise; requires only a small amount of data [96]; handles both discrete and continuous data [91].	Performs poorly when features are highly correlated; sensitive to data preparation method [96]; takes more runtime memory than SVM or simple LR, and is computationally intensive, especially if many variables [91].
PCA	Removes (correlated) features, to speed up the algorithm and reduce overfitting [106].	Data requires standardisation/scaling; components less interpretable than original features [106].
AdaBoost & variants	low generalization error; easy to code; computationally efficient, applicable to complex tasks; flexible, easily modified and easily combined with other learning algorithms [96].	Sensitive to outliers, may be noisy when training; - needs a large sample and can lead to "unwieldy" compositions [96]; may perform less well than tree-based algorithms for some data [107].

[**SM1.2**] *Input and output in ML methods used for EEG data analysis*

All types of EEG data features can be provided as input to [supervised] ML algorithms: whether time-domain, frequency-domain, complexity/nonlinear or entropic. Connectivity (spatiotemporal or network) measures can also be used. A useful, though the non-exhaustive, list is provided in [108]. Other features to be considered could include cordance [109], microstates [110], blink or other eye movement descriptors [111], spectral edge or centroid frequencies [112,113], the bispectral index [114], dimensionality [115] and LORETA results [116]. Feature selection will depend on the research area and the specific objectives of the tasks undertaken.

ML SVM or RF algorithm outputs may be categorical, in the form of a limited set of values, when used for classification, or a range of numerical values, when used for regression. LDA is used for classification, so its output is categorical, whereas this is strictly speaking not the case for PCA [117] or LR [118], which simply models the probability of output in terms of input.

Whatever the output, some form of metric is used to describe how well the algorithm has performed. The usual metrics for binary classification are receiver operating characteristic (ROC) curves, together with the area under the curve (AUC), as well as associated measures such as sensitivity, specificity, accuracy, and the Matthews correlation coefficient (MCC) [119,120]. Another option is to create a 'confusion matrix,' whose size depends on the number of classes to be obtained from the test set. The confusion matrix provides a useful summary of prediction results while at the same time casting light on any errors being made by the classifier, as well as the types of errors that are being made [82]. In his useful post on the topic, Brownlee includes a link to a confusion matrix online calculator made available by Marco Vanetti [121]. The calculator is simple to use and can accommodate multiclass classification, although there is no accompanying documentation. Cohen's *kappa* is often used as a metric for confusion matrices, comparing Observed Accuracy with Expected Accuracy (random chance). Unlike other metrics, such as Accuracy, the *kappa* value takes into account the possibility of chance agreement, and is therefore considered a more robust measure of a model's performance. *Kappa* can be used to evaluate a single classifier or to evaluate classifiers amongst themselves [122]. Vanetti's calculator provides *kappa* and overall accuracy as outputs. However, the F1-score (or F-measure) may also be more appropriate for imbalanced data than Accuracy [123], and for binary data the Matthews correlation coefficient may be more informative and useful than either F1 or Accuracy [119].

[**SM1.3**] *Algorithm hyperparameters in ML*

Algorithm hyperparameters are settings used to control an algorithm's behaviour [124] and are important to 'tune' for optimum performance. In one meta-learning study of six SVM hyperparameters, the most important were found to be 'gamma' (γ), the kernel density estimator, and 'complexity' (C), a constant that controls the trade-off between model simplicity and model fit. Of six hyperparameters for RF, the minimum samples per leaf and a maximal number of features for determining the split were the most important [125]. A useful list of hyperparameters, ensemble and other methods for most of the algorithms considered here is provided in papers on Auto-WEKA, one of the first automated ML systems, a software package for combined selection and hyperparameter optimisation of classification algorithms [63,126]. Another automated ML package that includes another such list, and that may outperform Auto-WEKA, is Auto-sklearn [127]. Auto-Keras is a corresponding package for Keras (see below), but for DL rather than ML (sometimes known as 'shallow') models. Auto-Keras has been claimed to outperform several traditional hyperparameter-tuning methods and state-of-the-art neural architecture search methods [128].

[**SM1.4**] *A rough sketch of DL methods used for EEG data analysis*

DL methods are necessarily unsupervised, so that hand-crafted feature extraction or feature selection are not needed [129]. The architecture of DL models is such that they not only contain input and output 'layers' (their basic building blocks), but also many others that are 'hidden,' with each layer taking information from previous layers and in turn passing information on to the next layer, so ultimately from input to output. When the training dataset is sufficiently large, DL enables automated feature selection, so that results can be highly accurate even without incorporating domain expertise [61].

The starting point to DL algorithms, and the simplest of them, is the MultiLayer Perceptron (MLP) [130], composed of layers of 'perceptrons', or linear binary classifiers, themselves modelled on biological neurons, with each neuron in one layer connected to every neuron in the next layer (so 'fully connected'). An MLP is thus an example of a feed-forward 'Artificial Neural Network' (ANN) [131], with more than one hidden layer [130]. A slightly less constrained (more general) form of the neural network, with not all layers necessarily fully connected, is known as a 'Deep Neural Network '(DNN) (already defined earlier). A useful – if technical – outline of DL algorithms, methods and applications is the DL textbook by Goodfellow *et al.* [124].

From the review papers located in PubMed on DL and EEG, the most frequently used DL algorithms appears to be Convolutional Neural Networks (CNN) [35,132] (257 hits), a natural extension to MLP in which 'convolution' (closely related to cross-correlation) is used instead of matrix multiplication in at least one layer [131], Long-Short Term Memory (LSTM) (76 hits), based on Recurrent Neural Networks (RNNs) [68,133] (50 hits) and DNN themselves (46 hits). RNNs are themselves an extension of Feed-Forward Neural Networks (FFNN) [130] (4 hits). Some reviewers considered that RNN performs significantly better on time series data while CNN is good for tasks like image classification [134] and is the most used method in several fields [132,135]. CNNs are not limited to 1-or 2-dimensional data, i.e., time-series or image data (there are even 2.5D and 7D versions of CNN [136,137]). After CNN, LSTM is the next most-used method in several reviews of DL for medical applications, followed by RNN and auto-encoders (AE) [135].

Deep Belief Networks (DBN), or Neural Networks (NNs) with multiple hidden layers, are another method [139] (9 hits), as are auto-encoders AE) (91 hits), often stacked (SAE) [132] (27 hits). An auto-encoder is an unsupervised learning algorithm that applies 'backpropagation,' with target values equal to the inputs (i.e., with both input and output layers having the same scale) [139]. Other algorithms

used include variants of Gradient Descent (GD) [91] or (cascaded) Restricted Boltzmann machine (RBM) [140], for example, but no EEG review papers located via PubMed included such methods.

Hybrid models may combine several different DL methods or a mix of ML and DL methods. Combining predictions from several models can be an elegant way to increase performance, increasing diversity among base classifiers. Ensemble models like bagging, boosting, and stacking are as possible in DL as in ML [141]. Multi-model Fusion Neural Networks (MFNN) are another possibility [139].

A frequent issue in EEG research is a low samples-to-features ratio, which can lead to overfitting, so severely reducing the effectiveness of DL and its generalisability. One way of overcoming this is to use 'data augmentation' (DA). There are many DA methods, but they can be divided into four main families – random transformation (e.g., warping and stretching), pattern mixing (e.g., interpolation, including SMOTE or Synthetic Minority Oversampling TEchnique), generative models (e.g., Generative Adversarial Networks (GAN) [142]) and decomposition (e.g., Independent Component Analysis, ICA) [143]. These are not all necessarily appropriate for all types of DL. For instance, the hybrid LSTM-CNN, also known as an LSTM-FCN (Long-Short Term Memory Fully Convolutional Network) was found in one review to react poorly to DA, particularly when the number of patterns per class is small. MLP (Multilayer Perceptron) also did not respond well but to a lesser extent. Window warping and slicing were the most generally useful DA methods [143].

DA is increasingly used in EEG research, with Sliding window (SW) being the most common method, although with no consensus as yet on the best overlapping percentage to use between consecutive windows. Noise addition and GAN were also frequently used, and all three methods provided useful improvements in accuracy. Resampling approaches were less useful [144].

In the papers reviewed by Lashgari et al., the Sliding window method was used with both CNN and LSTM, GAN with CNN, SVM and others, and Noise addition with CNN, LSTM, and their combination, as well as with SAE.

Keras is an application programming interface (API) linked to TensorFlow, one of the major frameworks used in DL. Both Keras and Tensorflow itself are very commonly used [135]. TensorFlow was given the top rating for framework performance in one review [139].

DL methods require some form of 'accelerator' to reduce power consumption and increase processing speed because they are computationally demanding. This can take the form of a dedicated central processing unit (CPU) or Graphics Processing Unit (GPU) capable of parallel processing. The latter is more economical to run [139].

Some advantages and disadvantages of the main DL algorithms used in EEG research are listed in **Table S2**.

Table S2. Some advantages and disadvantages of the main DL algorithms used in EEG research. (See Abbreviations list in paper for interpretation.)

Algorithm	**Some advantages**	**Some disadvantages**
CNN	Hand-crafted feature selection not required; robust against noise; does not need a feature extraction step, as can self-learn from input data [129]; pre-processing may not be needed [130]; invariant to transformations [145]	Requires a lot of data to learn; computationally expensive [129], as many test runs may be needed to tune parameters [146]; performance highly dependent on hyperparameters [35]; often called 'black boxes', as it is not easy to understand what goes on inside the

		model [147]; in general, notoriously difficult to interpret [148]; unsupervised, so may not achieve accurate segmentation; may not handle heterogeneous data well [135]
LSTM	Can process temporal or sequential information [74]; bidirectional LSTM (allowing both forward and reverse dependencies) may outperform standard LSTM, with fewer parameters required and thus fewer computing resources [149]; for some data, stacked LSTM may outperform LSTM or bidirectional LSTM [135].	As with other DL methods, training can be time-consuming [149]; sensitive to recording frequency [150]; may fail in the absence of obvious structure in input data [151]; huge memory requirements for network parameters [1152]; input data length has to be truncated or padded to a constant, finite number [153]; dropout difficult to implement; sensitive to different random weight initialisations [135].
RNN	Well-suited for time-series or sequential data; performs well on data with long-term dependency [154]; bidirectional DL is possible; accurate [135]; identical parameters may be used across all steps [1155].	Complex and computationally intensive [135]; not easy to track long-term dependencies (gradient decay over multiple layers) [155].
DNN	Can be pre-trained as a DBN; efficient in representing high dimensional and correlated features [156]; able to cope with nonlinear relationships [157].	Computationally expensive to train, and to determine the training method and the hyperparameters [130]; may be difficult to interpret [158].
ANN	High tolerance to noisy data [94].	computational burden, prone to overfitting (requires long training time with large datasets), 'Black box' [94].
AE	It may be robust, with high accuracy and good scalability [135].	Liable to overfitting in its simplest form [159]; may become ineffective if errors are present in first layers [145]; training time may be high 155].

[**SM1.5**] *Input and output in DL methods used for EEG data analysis*

Power spectral density (PSD), wavelet decomposition, and statistical measures of the signal (e.g., mean and standard deviation) are the three most common input formulations used in the studies reviewed by Craik et al. Nineteen of those used a wavelet transform, only 2 wavelet decomposition, and 6 used PSD [132]. In a review on Data Augmentation (DA) [144], raw time-domain signals were used in 36% of studies, and features calculated from the raw signals (the most common approach) in 49%). Spectrograms processed as images were used in 15%.

For some methods (such as CNN) EEG data can be used 'raw' (unprocessed), with artefacts left in [130]. Otherwise, artefacts can be removed manually or algorithmically. Useful tables in the review paper by Craik et al. [132] indicate that, where the method was described, manual artifact removal (MR) was most common for SAE (3 of 7 studies), DBN (5 of 10) and Hybrid (4 of 10), with algorithmic removal (AR) often used for RNN (4 of 9). For CNN, 11 of 37 studies used MR, 9 used no removal and 1 used AR Many studies did not report the method used.

Craik et al. [132], in their useful review of DL for EEG data, indicate that numbers of convolutional layers in CNN models are generally between 1 and 7, although as many as 16 have been used, while between 1 and 6 fully connected layers and between 2 and 7 output layers are found.

[**SM1.6**] *Algorithm hyperparameters in DL*

In DL, hyperparameters such as learning rate, number of hidden layers in a neural network and regularisation strength are the variables that determine the architecture and behaviour of a model. Hyperparameter optimisation or choosing the best values for these variables in order to improve model performance (accuracy and speed), can be done using a variety of methods (e.g., grid search, random search or Bayesian optimisation).

However, the parameter tuning process can be time-consuming and may require significant computational resources. Effects on data preparation, model building and training will also depend on the specific algorithms and techniques used. It is therefore important to consider carefully which hyperparameters to optimise and how to go about optimizing them in order to obtain the best possible results.

There are now several algorithms available for hyperparameter optimisation, including several based on stochastic GD, such as Adam (Adaptive momentum estimation), a combination of GD with Momentum and Root Mean Square Propagation (RMSprop) [160]. Adam has been suggested as the default optimisation algorithm for DL training [79] and works well with the little tuning of hyperparameters except for learning rate [160], performing well during both testing and training phases of DL [161]. However, in some situations, Adam may be outperformed by Nadam (Nesterov-Accelerated Adaptive Moment Estimation) in terms of stability, convergence rate, training speed and performance [162]. Some advantages of Adam are listed in [163]. For CNN, performance is highly dependent on hyperparameters such as the number of convolution layers, the size and number of kernels, the size and type of pooling windows used, and stride size. However, there is no specific strategy to choose their values, so performing a large number of iterations is the only way to determine the best values of the hyperparameters.

For both ML and DL algorithms, 'fusion' methods may enhance performance. Fusion may occur at the data, feature, or decision level [164].

[SM2] The CNN-LSTM hybrid mode

[**SM2.1**] *A brief description of CNN*

The CNN model, developed originally for use with 2-D images, performs well on 1-D data such as the EEG. CNN usually includes three types of layers:

(1) **Convolutional layers**, with learnable filters (kernels) that slide across pre-processed signals to extract local features from the input [165]. 'Stride,' or the step size of the filter sliding over the input [166], controls how the filter shifts around the input signal. A feature map results after repeated overlapping applications of the filter in the convolution step [167].

A 1-dimensional convolution layer (Conv1D) creates a convolution kernel that is convolved with the layer input over a single spatial dimension, without 'flattening'. The input data for Conv1D may be padded (as mentioned above for LSTM) [168]. Weights are shared over different layers, which can help reduce parameters, speed up convergence to a solution and avoid overfitting [169].

L2 regularisation (or ridge regression) in the convolution layer may also reduce overfitting, encouraging weight decay in sequential learning algorithms [170].

(2) **Pooling, or down-sampling layers,** to reduce dimension to prevent overfitting and decrease computational demand. Average pooling and Max pooling are two common methods used [75].

A CNN includes alternating convolutional layers followed by pooling layers 165].

(3) One or more **fully connected layers**, using an Activation function to introduce nonlinearity into the output [154].

The Activation function

DL methods employ an 'activation function' that defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. Most real-world data is not linearly separable; without introducing nonlinearity in some way, nontrivial classification would not be possible. Commonly used activation functions used in hidden layers are Rectified Linear Activation (ReLU), Logistic (Sigmoid) and Hyperbolic Tangent (Tanh). Nowadays, CNN and MLP mostly use ReLU [171], which makes for faster training than the Sigmoid or Tanh methods [165]; RNN (and LSTM) mostly use Sigmoid activation for recurrent connections and the Tanh activation for output. Output activation functions (used in the output layer) may be linear (or 'no activation,' used for regression problems), or Sigmoid or SoftMax (used for binary/ classification and multiclass classification, respectively) [172]. Of the 37 CNN studies analysed by Craik et al. [132], 22 used ReLU for the convolutional layers, and 27 used SoftMax for the final fully-connected layer. With SoftMax, the output can be interpreted as probabilities [173].

In Keras, a layer is most often also 'dense,' with each neuron (network point, node) in the layer receiving input from all neurons in the previous layer [174].

In Keras (or TensorFlow), models may be built using three different methods: Sequential API (application programming interface), Functional API or Model subclassing. The Sequential method is the simplest, allowing the creation of models layer-by-layer in a step-by-step fashion, but this is also a limitation. It does not allow models to share layers, or to have multiple inputs or outputs [175].

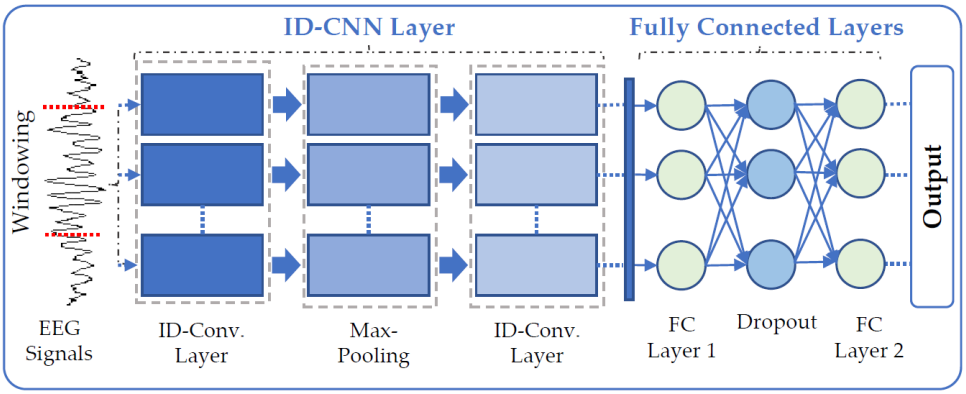

Figure S3. Generic CNN architecture (adapted from Shoeibi et al., 2021 [176] by Tony Steffert).

[SM2.2] *A brief description of LSTM*

The LSTM is a subtype of RNN that overcomes the 'vanishing gradient' problem inherent in RNNs (but not necessarily the 'exploding gradient' of a nonconverging solution). 'Vanilla' LSTM is the most commonly used version of LSTM, with hidden layers containing LSTM 'blocks' that features three gates (input, forget, output), a single 'cell' (the Constant Error Carousel), an output activation function, and peephole connections within the block (Figure S4). The output of the block is recurrently connected back to the block input and all of the gates. The 'forget gate' and the output activation function are LSTM's most critical components. learning rate and network size are the most crucial tuneable LSTM hyperparameters [177]. There are several subtypes of LSTM, including with or without a Forget Gate or 'Peephole connection,' stacked or bidirectional LSTM networks and multidimensional LSTM networks, and even convolutional LSTMs (ConvLSTM) [140]. However, Vanilla LSTM is likely to perform almost as well as variants [177]. More information on LSTM can be found, for instance, in Greff *et al.* 2017 [177].

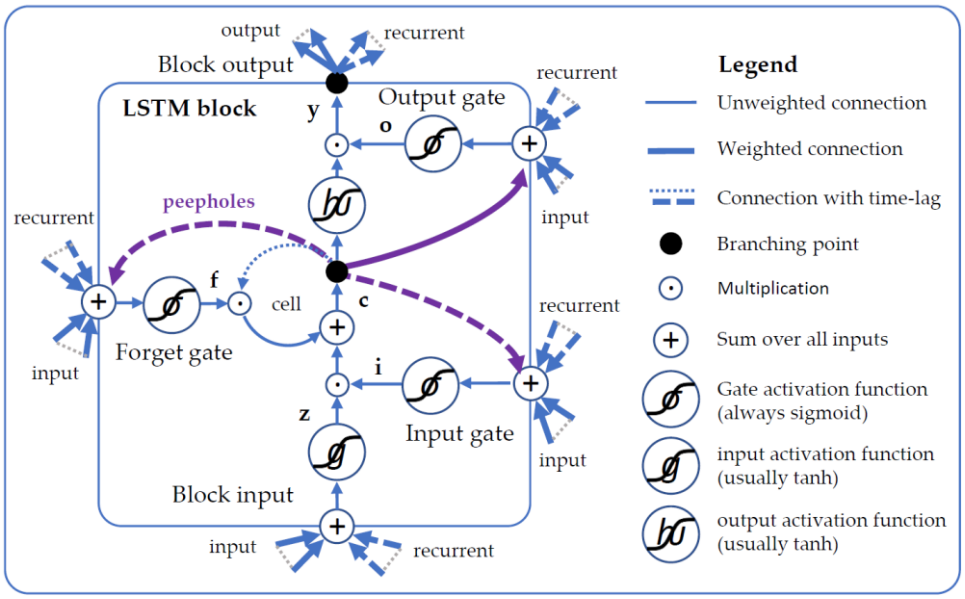

Figure S4. Generic LSTM block, showing input, output and forget gates, a single 'cell' (the Constant Error Carousel), an output activation function, and peephole connections within the block (adapted from Greff et al., 2017 [177] by Tony Steffert).

More on algorithm architecture

For both CNN and LSTM, regularisation may be used to minimise overfitting, and in Keras, this is often done with 'Dropout,' a method that can be applied to convolutional, dense fully connected or recurrent layers, among others. During the training of the model, some of the layers' nodes (or input units are dropped randomly, so that the model can simulate having many different network architectures. Rather like the addition of random noise in DA, this can improve performance, especially where there is a limited amount of training data [178]. 'Recurrent dropout' may be combined with this 'conventional forward dropout' to further improve results using LSTM [179].

Once a model is created in Keras, it is configured (compiled) by setting various hyperparameters such as a number of epochs and batch size, the 'loss function' (or cost function) to be optimised, and the method of optimisation to be used (see above, Section 1.6). The loss function distils all aspects of the model down into a single number that then allows candidate solutions to be ranked and compared [171]. Backpropagation is an efficient method that may be used to compute the gradient of the loss function concerning a model's parameters (weights) [124, p 259] and can be used to train NNs in conjunction with (stochastic) gradient descent [79].

The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset, while batch size controls the number of training samples to work through before the model's internal parameters are updated [167]. The loss function will vary, depending on the purpose of the model. Common loss functions include mean square error (for regression), binary cross-entropy (for binary classification) and categorical cross-entropy for multi-class classification [180]. As a loss function, cross-entropy is likely to perform better than traditional square error methods [180]. Cross-entropy is sometimes calculated from the related Kullback-Leibler divergence (relative entropy) [180].

Figure S5 shows a hybrid CNN-LSTM model, similar to that used in this study.

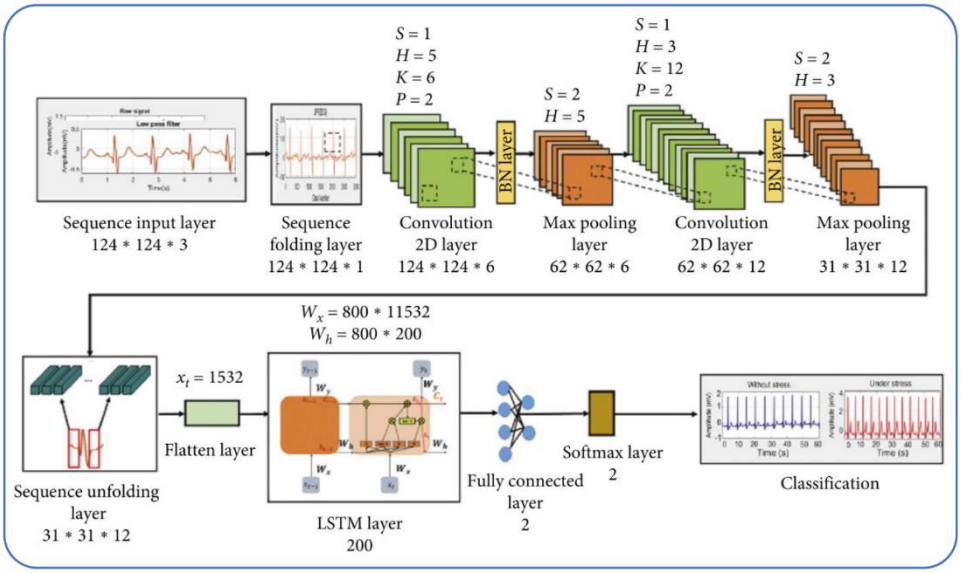

Figure S5. Hybrid CNN-LSTM model, as used in this study, but without the Flatten layer (adapted from Kang *et al.* 2021 [181] by Tony Steffert).

References

For references, see the main paper:

Uyulan, Ç., Mayor, D., Steffert, T., Watson, T., Banks, D. Classification of the central effects of transcutaneous electro-acupuncture stimulation (TEAS) at different frequencies: A Deep Learning approach using wavelet packet decomposition with an entropy estimator.