Comparative Analysis of Deep Neural Network Models for Time Series Classification

Kieran Molloy, 35762970

***Abstract*—The implementation of automatic Electrocargiogram (ECG) classification poses massive real world benefits, allowing a more streamlined patient diagnosis pathway for heart conditions that would not be found without cardiologist input. ECG Classification is a complex time series problem, and there are many machine learning solutions that can be used to analyse and classify ECG data - these are mostly hand-crated heuristics or feature selected. This paper poses five deep learning time series classification architectures that can classify ECG data swiftly and with increased performance than an average cardiologist, furthermore using Wilcoxon-Holm post-hoc analysis to determine if a set of classifers are statistically significantly different.**

***Index Terms*—DNN, CNN, TSC, MLP, RESNET, RNN, MCNN, LENET**

I. Introduction

WITHIN the previous ten years Deep Learning has made significant strides and Time Series Classification poses one of the largest challenges. With temporal data becoming widely available, there has been a significant increase of algorithms proposed recently [1]. With temporal data requiring the order to be maintained, the time series classification problem is encountered in many scenarios, such as ECG - as is the main focus within this report, acoustic classification and cyber-security. Most general heart problems can be diagnosed from ECG signals, and with hundreds of millions of these recorded annually, physicians are not able to manually review and diagnose conditions. This motivates the use of Machine Learning approaches to swiftly and accurately diagnose these heart irregularities for further review by a physician. The main disadvantages of machine learning approaches is the inability to find the most appropriate features that give high classification accuracy. Deep Learning architectures can resolve this problem with this paper focusing on the end-to-end deep learning models, and implementing a series of models on selected ECG datasets. The results are investigated using Critical difference diagrams - leveraging Wilcoxon and Friedmann tests, and evaluating classification metrics.

II. Background

ECG Classification is a difficult problem, and has had significant attention from the medical community and increasingly computer scientists that specialise in machine learning approaches. Recently a 34-layer convolutional neural network was capable of exceeding the average cardiologists performance in precision and recall [2]. Other works present binary ECG classification algorithms that use manual heuristics or feature engineering, these work well in single scenarios but cannot be transitioned to solve other ECG classification problems, therefore it is of interest to define a general model for ECG classification that is able to be translated to all ECG problems. The ECG data which can be obtained from patients is time series data, often a duration of around 45 seconds and a sampling time of 0.03s. As stated in the introduction it is desireable to remove the feature selection aspect of these classification algorithms, as that requires expert and subject specific knowledge. Therefore this paper will discover what deep learning approaches have been succesful for other time series classification problems and translate those architectures to the ECG classification problem.

Deep learning approaches for Time Series Classification (TSC) are separated into two key groups [3];

- generative
- discriminative

These are further separated into sub-groups, shown in Figure 1

A. Generative Models

Generative models often have an unsupervised training step that precedes the learning phase of the classifier [3]. This type of network is known as a model-based classifier [4]. Some of these approaches include auto-regressive models [5], hidden markov models [6] and kernel models [7]. The goal for generative models is to create/find a representation of time series prior to training a classifier [1], [3], , usually to model a time series, classifiers are preceded by an unsupervised pre-training phase such as stacked denoising auto-encoders (SDAEs) [8], [9]. A generative CNN-based model was proposed in [10]; [11] where the authors introduced a deconvolutional operation followed by an upsampling technique that helps in reconstructing a multivariate time series. Deep Belief Networks (DBNs) were also used to model the latent features in an unsupervised manner which are then leverages to classify univariate and multivariate time series [12], [13]. In [14], [15], [16], an RNN auto-encoder was designed to first generate the time series then using the learned latent representation, they trained a classifier (such as SVM or Random Forest ensemble method) on top of these representations to predict the class of a given input time series. Other studies such as [17], [18], [19], [20] used self-predict modelling for time series classification where Echo State Networks were first used to re-construct the time series and then the learned representation in the reservoir space was utilised for classification, they have been used to define a kernel over the learned representation followed by an SVM or MLP classifier [7], [21], [22]. Other papers have presented a meta-learning evolutionary-based algorithm to construct optimal architecture for univariate or multivariate time series [10], [23].

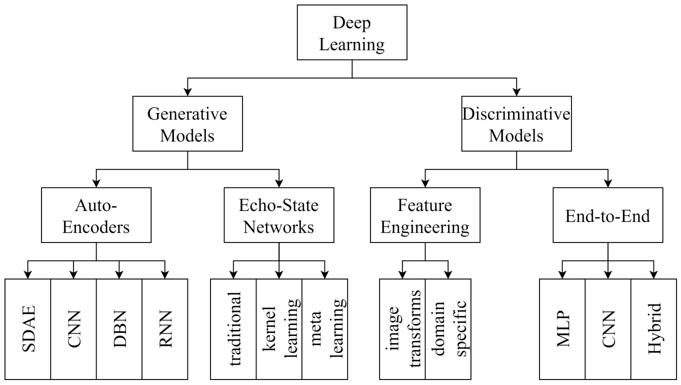


Fig. 1: Hierarchy of deep learning approaches for time series classification from [1][3]

B. *Discriminative Models*

A discriminative deep learning model is a classifier, or regressor, that directly learns the mapping between an input time series and outputs a probability distribution over the class variables in a dataset. The most frequently applied feature extraction method for

III. Methodologies

Formally, A univariate time series $X = \{x_1, x_2, \ldots, x_T\}$ is an ordered set of real values, where the length of X is equal to the number of real values T. Naturally, an M-dimensional time series, MTS, can be defined as $\bar{X} = [X^1, X^2, \ldots, X^T]$ which consists of M different univariate time series where $X_i \in \mathbb{R}^T$.

A deep neural network is a composition of parametric functions, known as layers, where each function is a representation of the input domain [24]. Each layer l_i contains any integer number of neurons, which are independent compute units that return the layer output, traditionally these also apply a non-linearity function such as relu or sigmoid. These layers are chained where layer l_i takes input from layer l_{i-1} and returns its output to layer l_{i+1}, these transformations are controlled by θ_i, otherwise known as weights, which link the layers together. Hence, given an input dataset x, a neural network can be defined as

$$f_L(\theta_L, x) = f_{L-1}(\theta_{L-1}, f_{L-2}(\theta_{L-2}, \ldots, f_1(\theta_1, x))) \quad (1)$$

where f_i is the non-linearity function for layer i. This process is known as feed-forward propagation.

A. *Multi-Layer Perceptron*

A Multi-Layer Perceptron (MLP) constitutes the simplest form of a deep learning model, the architecture of the model is known as fully connected - since the neurons in layer l_i are connected to every neuron in layer l_{i-1} where there are $i \in [1, L]$ layers, each of these connections hold a weight. A general form of an MLP applied to a time series X

$$A_{l_i} = f(w_{l_i} \cdot X + b) \quad (2)$$

with w_{l_i} being the set of weights, b the bias term and A_{l_i} the activation function of the neurons in layer l_i. A problem with MLP's is the inability to exhibit any spatial invariance, so temporal information is lost and each element is considered independently. The final layer takes the form of a probabilistic distribution over the class variables, often a softmax or sigmoid, and has equal neurons to the number of classes in the dataset. The weights presented in equation 2 are learned automatically using an optimisation algorithm that minimises an objective function - generally the Adam optimiser is used throughout this paper. For an optimiser to approximate the error of values, a loss function that can quantify this error needs to be defined, a commonly used loss function is categorical cross entropy:

$$L(X) = -\sum_{j=1}^{K} Y_j \log \hat{Y}_j \quad (3)$$

where L is the loss of time series X. Naturally, the average loss of the whole training set is defined by:

$$J(W) = \frac{1}{N} \sum_{n=1}^{N} L(X_n). \quad (4)$$

Hence, the loss function minimises the learned weights w using a gradient descent method:

$$w = w - \alpha \frac{\partial J}{\partial w} \bigg| \; \forall \; w \in W \quad (5)$$

where α is the learning rate of the optimisation algorithm. This model is capable of auto-tuning the parameters w in order to search and find a local minimum J. The partial derivative cannot be directly computed with respect to a certain parameter w, the chain rule of derivative is employed which is in fact the main idea behind the backpropagation algorithm [25].

B. *Convolutional Neural Networks*

The idea of using artificial neural networks for processing images was proposed in [26] which automated the recognition of numbers, leading to the development of the widely used MNIST dataset today, this paper proposed an architecture now known as Convolutional Neural Networks (CNN). A CNN consists of input and outputs layers, as well as hidden layers, like the MLP described previously. The hidden layers consist of convolutional layer, pooling layer and a fully connected classifier - which is a perceptron based upon the features obtained on previous layers.

A convolution can be seen as applying a sliding filter over a time series. Unlike the 2-dimensional variant which have width and height, the filters exhibit a single dimension only which represents time. The filter is synonymous with a generic non-linear transformation of the time series - if the time series is convoluted with a filter of size 3 with a univariate time series,

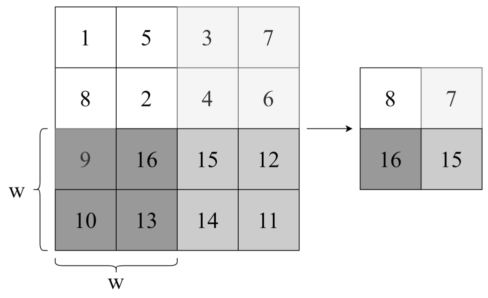


Fig. 2: The principle of operation of the pooling layer that implements the max pooling operation on a 2-dimensional array

by setting the filter to $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, the convolution will result in applying a moving average with a sliding window of length 3. A general form of this for any centered time step t:

$$C_t = f(w \cdot X_{t-\frac{l}{2},t+\frac{l}{2}} + b) \mid \forall\ t \in [1, T] \tag{6}$$

where C is the convolution, the dot product of time series X and filter w with lengths T and l respectively, the bias parameter b and a non-linear function f such as ReLU. Applying several filters on a time series will result in a multivariate time series whose dimensions are equal to the number of filters used. An intuition behind applying several filters on an input time series would be to learn multiple discriminative features useful for the classification task. Unlike the previously discussed MLP's, the same convolution will be used to find the result for all time stamps t. This is known as weight sharting and is a powerful property of the CNN's which enable filter learning with no invariance across time. When considering a multi-variate time series as input to a convolutional layer, the filter is adapted to have dimensions equal the the time series instead of the ordinary single dimension, t, time. Weights are learned automatically, since they highly depend in the targeted dataset and its attributes, the optimal filter is custom to these, where optimal is a filter that enables the classifier to easily discrimate between the classes - generally, when applied, this takes the form of a pooling operation, followed by a discriminative filter, followed by the classifier. Local Pooling, such as average or max, with max pooling demonstrated in Figure 2, takes an input time series and reduces it length T by aggregating over a sliding window of the time series. For example, if the sliding windows length is 4, the resulting pooled time series with have length $\frac{T}{4}$ (where $T \mod 4 = 0$). In addition to pooling, some architectures include layers that normalise to increase convergence speed, for time series data, the batch normalisation operation is performed over each channel, thus preventing a covariate shift across a batch of time series [27]. An alternative approach, similar to z-normalisation, is to normalise each instance instead of batch, learning the mean and standard deviation of each instance for each layer via gradient descent [28]. The final discriminative layer takes the result of the convolutions and returns a probability distribution over the classes, this layer is usually a softmax operation. In order to train the network, and learn the parameters, the process is the same as the MLP described above, a forward pass, followed by back-propagation [25]. An example of a CNN architecture for time series classification with three convolutional layers is illustrated in Figure 3.

C. Echo-State Networks

A further extension of deep learning models is the Recurrent Neural Network (RNN). This method is rarely used with time series data, with the exception of time series forecasting, due to three key factors.

- the architecture is designed to predict an output for each element, so for time series, each time stamp [3]
- the models have a vanishing gradient problem when training on long time series [29]
- the models are hard to train and parallelise [29]

These problems led to the repurpose of Echo State Networks (ESNs) [30], which were first created for time series prediction in wireless communication channels [31]. They alleviate the vanishing gradient problem by eliminating the need to compute the gradient for hidden layers, also reducing the training time. These hidden layers are initialised randomly, and is known as the *reservoir*, the core principal of an ESN, which is sparsely connected to an RNN. Consider an ESN with input dimensionality M, neurons in the reservoir N_r, and output dimensionality K. Letting $X(t), I(t), \hat{Y}(T)$ denote the multivariate time series, hidden state and output activity for time t respectively. Formally the hidden state is:

$$I(t) = f(W_{\text{in}}X(t) + WI(t-1)) \mid \forall\ t \in [1, T] \tag{7}$$

where f, again, is the activation function, but usually ESNs use elementwise $\tanh$. With the output calculated by

$$\hat{Y}(t) = W_{\text{out}}I(t) \tag{8}$$

thus classifying each time series element. Figure 4 demonstrates an example ESN for univariate time series classification, to be classified into k classes.

D. Approaches

The approaches this analysis will consider are presented in Table I, and they are further described in Table II.

IV. Experimental Setup

A. Datasets

This paper considers 4 datasets, shown in Table III, 3 of these are from the UCR archive [32] which contains 85 univariate time series datasets. The 4th dataset, which is also the key dataset of this report, is a subset of ECG5000

1) ECG1: This dataset is provided by TensorFlow and their method of construction is unclear. As described previously, this is the key dataset and a subset of ECG5000 where the classes have been modified from 5 classes provided by automated annotation to 2 classes, corresponding to abnormal rhythm or normal rhythm. This is provided as a single entity which must be split into test and train

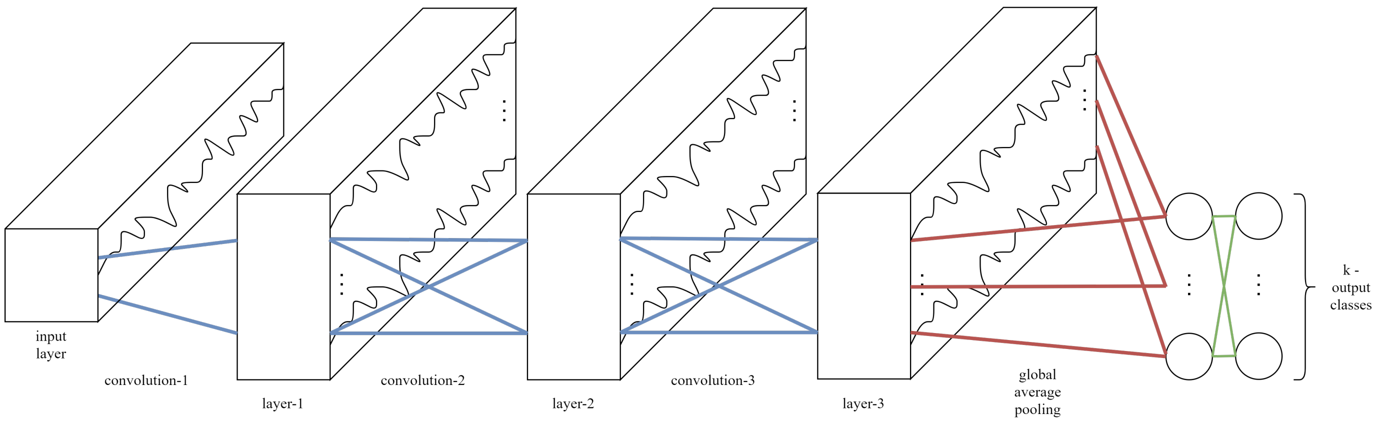


Fig. 3: Fully Convolutional Neural Network Architecture for Time Series Classification

TABLE I: Method Architecture's hyperparameters for each method

Methods	Architecture							
	Layers	Conv	Invar	Normalise	Pooling	Feature	Activate	Regularise
MLP	4	0	0	None	None	FC	ReLU	Dropout
ResNet	11	9	10	Batch	None	GAP	ReLU	None
Encoder	5	3	4	Instance	Max	Att	PReLU	Dropout
MCNN	4	2	2	None	Max	FC	ReLU	None
t-LeNet	4	2	2	None	Max	FC	ReLU	None

TABLE II: Optimisation hyperparameters for each method

Methods	Architecture						
	Algorithm	Valid	Loss	Epochs	Batch	Learning Rate	Decay
MLP	AdaDelta	Train	Entropy	500	16	1.0	0.0
ResNet	Adam	Train	Entropy	500	16	0.001	0.0
Encoder	Adam	Train	Entropy	100	12	0.00001	0.0
MCNN	Adam	Split 20	Entropy	500	256	0.1	0.0
t-LeNet	Adam	Train	Entropy	500	256	0.01	0.005

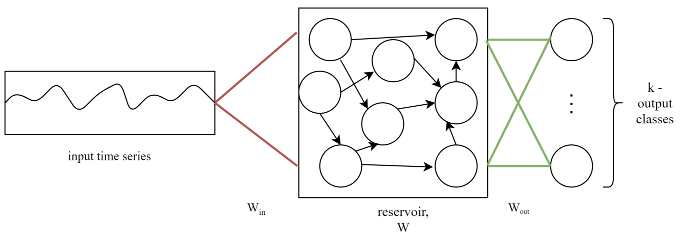


Fig. 4: Echo State Network Architecture for Time Series Classification

TABLE III: Dataset Properties

Dataset	Properties			
	Train Size	Test Size	Length	Classes
ECG1	4500	500	140	2
ECG200	100	100	96	2
ECG5000	500	4500	140	5
ECGfivedays	23	861	136	2

2) ECG200: The dataset is from a PhD submission attempting to generalise feature extraction for strucurtal pattern recognition, [33], Each series traces the electrical activity recorded during one heartbeat, and the 2 classes represent a normal heartbeat and a myocardial infarction. The properties are again described in Table III

3) ECG5000: The original dataset for this is a 20-hour long ECG from PhysioNet, where it is from BIDMC Congestive Heart Failure Database (chfdb) and it is record "chf07". It was originally published in [34]. The data was pre-processed by [35] in two steps (1) extract each heartbeat, (2) make each heartbeat equal length using interpolation. After this, 5000 heartbeats were randomly selected, the patient is known to have severe congestive heart failure and class values were obtained by automated annotation.

4) ECGfivedays: This dataset is from [35], in addition to ECG5000 and subsequently ECG1, it is data from a 67 year old male where the two classes are the date of recording, namely, 12/11/1990 or 17/11/1990 - it is nondescript what these dates represent.

B. Pre-Processing

All datasets have their labels extracted and the data is coerced into numeric form, with the labels, using sklearn one-hot encoding, converted to one hot vectors and an additional variable which stores the argmax of these vectors. Univariate datasets are then changed into multivariate problems with one dimension - this is then passed to the respective classifier.

C. Experiments

For each dataset, the 5 deep learning models (as in the previous section) are trained with 3 runs each. Each run uses the same test and train subsets, as this is done either externally in the case of datasets from UCR, or internally via a seeded test-train split, in the case of ECG1. Each run will generate different initial random weights, and so the 3 runs are averaged to reduce bias. In total there are 20 models, with 60 experiments (not including MCNN hyper-parameter searching). This was run on an Nvidia GTX 970, with 4GB of GDDR5. The total sequential running time was approximately 27 hours. Each method was implemented using the open source deep learning library Keras [36], with TensorFlow [37] backend. Using the mean accuracy over the 3 runs [38], the Friedman test is used to reject the null hypothesis then a pairwise post-hoc analysis - where the average rank comparison is replaced by a Wilcoxon signed-rank test with Holm's alpha correction. This is visualised using critical difference diagrams, where a thick horizontal line shows a group of classifiers that are not-significantly different in terms of accuracy.

V. Results

All experiments from the previous section were compiled into a single table from which all analysis is performed and selected via R, except the critical difference diagram creation which is performed in python [1]. The critical difference diagram for all 4 datasets is shown in Figure 5. There are 3 bands of classifiers, the Encoder and ResNet outperform all other models with average ranks of 1.2917 and 1.7500 repectively. These classifers are consistently ranked 1st and 2nd for all datasets and significantly outperform the FCN architecture, which contrasts previous results where FCN has been found to outperform ResNet in most scenarios - this is reinforced in the validation of larger archives supporting this [1]. The middle band consists of MLP and t-LeNet, with average ranks of 3.7500 and 3.3333 respectively - MLP is not a widely used method as it has been improved to give increased accuracy at decreased computation, it is however, an important baseline method. The t-LeNet architecure outperformed expectations by being the middle method, previous experimentation has shown that the t-LeNet suffer from low accuracy problems with time series data due to their time slicing-majority voting scheme classification method. Additionally MCNN suffers from this problem, and was expected to perform somewhat poorly.

The primary dataset of this comparative analysis is *ecg1*, the critical difference diagram is shown in Figure 6 - it returns similar results to the all-datasets version in Figure 5 - with the notable exception that there is no statistical significance between the results. This is because all methods are able to quickly gain an accuracy ≈ 1.000, but it must be added that all models see their peak performance in the last ≈ 10 epochs, their accuracies can be seen in Figure 9, these also have low loss values, seen in Figures 13, 14, in addition to good precision and recall values, however notably Encoder scores the best, followed by ResNet and MLP - whom score very closely - with t-LeNet attaining the lowest scores and also largest error margins. All the error margins for *ECG1* can be attributed to having a smaller test set, as well as the random starting point. This is not the case for *ECG200*, which has a far larger test case, with the error margins for accuracy, precision and recall shown in Figure 10, t-LeNet again scores very poorly, with the other models performing admirably, and Encoder returning the strongest results. The critical difference diagram for *ECG200* could not be created due to results being within a too small error margin. The critical difference diagram for *ECG5000* is shown in Figure 7 and returns the same as the previous two with Encoder and ResNet performing the strongest however, as shown in the thick black line, the difference is statistically insignificant, the classification metrics are in Figure 11 where high accuracy are seen for all methods, however very low precision and recall values suggest this model is over-fitted, as well as large error margins for the precision metric. There is similar to *ECGfivedays* which sees Encoder and ResNet attaining extremely strong scores of ≈ 1 for accuracy, precision and recall, as seen in Figure 12. t-LeNet and MLP are very weak, having low accuracy and low recall, however according to the critical difference diagram, in Figure 8, the difference is statistically insignificant.

VI. Conclusion

This analysis presented recent successful architecures for Time Series Classification, it was described how Deep Neural Networks are categorised and selected methods were mathematically defined. Five end-to-end methods were implemented , training them with 4 ECG datasets - 1 primary and 3 secondary datasets. The results show that Encoder and ResNet produce great results and outperform previous attempts to use deep learning to classify ECG data, with Encoder scoring an average accuracy of 96%, additionally, this accuracy is far above the average cardiologist [2]. Although this comparative analysis extensively analysed methods and ECG data, further study is required regarding data augmentation, and transferability of learning. Additionally an extensive study of computation time, and expensive functions is required to enable these methods to process more data efficiently. In conclusion, with ECG data becoming more easily shared within healthcare systems, leveraging deep learning architectures capable of learning from massess of data with ease, deep learning is prime position to enable healthcare providers to implement their own analysis to automatically recognise heart conditions that cardiologists do not have the time to evaluate.

Acknowledgment

I would like to thank Hassan Ismail Fawaz, whose work on deep learning is the foundation upon which this report is largely based [1].

References

[1] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

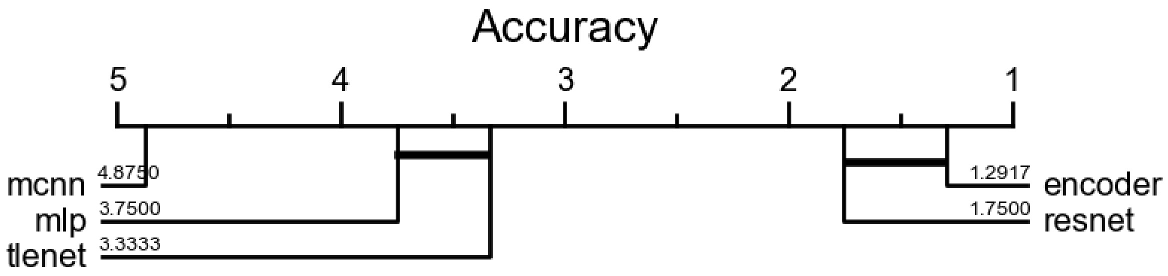


Fig. 5: Critical Difference diagram showing pairwise statistical difference comparison of five deep learning classifiers on all datasets

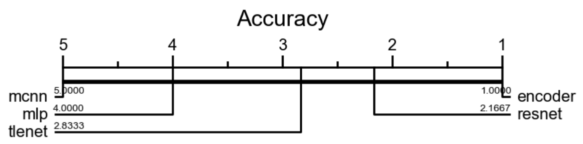


Fig. 6: Critical Difference diagram showing pairwise statistical difference comparison of five deep learning classifiers on the dataset *ecg1*

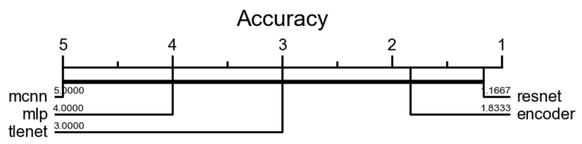


Fig. 8: Critical Difference diagram showing pairwise statistical difference comparison of five deep learning classifiers on *ecgfivedays*

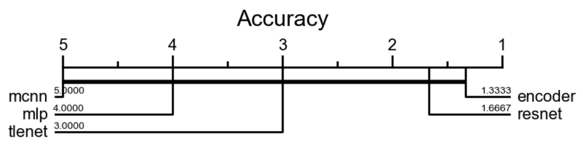


Fig. 7: Critical Difference diagram showing pairwise statistical difference comparison of five deep learning classifiers on *ecg5000*

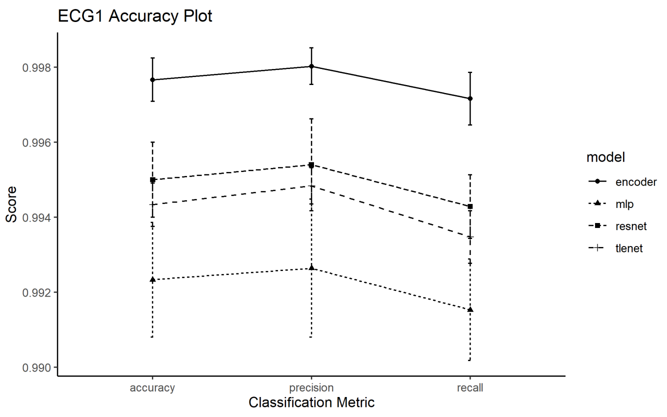


Fig. 9: Classification Metics (with average, sd error bars) for each model for *ECG1*

[2] P. Rajpurkar, A. Y. Hannun, M. Haghpanahi, C. Bourn, and A. Y. Ng, "Cardiologist-level arrhythmia detection with convolutional neural networks," 2017.

[3] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, pp. 11–24, 2014.

[4] A. J. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version," *CoRR*, vol. abs/1602.01711, 2016. [Online]. Available: http://arxiv.org/abs/1602.01711

[5] A. J. Bagnall and G. Janacek, "A run length transformation for discriminating between auto regressive time series," *Journal of Classification*, vol. 31, pp. 154–178, 2014.

[6] A. Kotsifakos and P. Papapetrou, "Model-based time series classification," in *IDA*, 2014.

[7] H. Chen, F. Tang, P. Tino, and X. Yao, "Model-based kernel for efficient time series analysis," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, Aug. 2013. [Online]. Available: https://doi.org/10.1145/2487575.2487700

[8] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 899–907.

[9] Q. Hu, R. Zhang, and Y. Zhou, "Transfer learning for short-term wind speed prediction with deep neural networks," *Renewable Energy*, vol. 85, no. C, pp. 83–95, 2016. [Online]. Available: https://ideas.repec.org/a/eee/renene/v85y2016icp83-95.html

[10] Z. Wang, W. Song, L. Liu, F. Zhang, J. Xue, Y. Ye, M. Fan, and M. Xu, "Representation learning with deconvolution for multivariate time series classification and visualization," *CoRR*, vol. abs/1610.07258, 2016. [Online]. Available: http://arxiv.org/abs/1610.07258

[11] R. Mittelman, "Time-series modeling with undecimated fully convolutional neural networks," 2015.

[12] S. Wang, G. Hua, G. Hao, and C. Xie, "A cycle deep belief network model for multivariate time series classification," *Mathematical Problems in Engineering*, vol. 2017, pp. 1–7, 2017. [Online]. Available: https://doi.org/10.1155/2017/9549323

[13] D. Banerjee, K. Islam, G. Mei, L. Xiao, G. Zhang, R. Xu, S. Ji, and J. Li, "A deep transfer learning approach for improved post-traumatic stress disorder diagnosis," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, Nov. 2017. [Online]. Available:

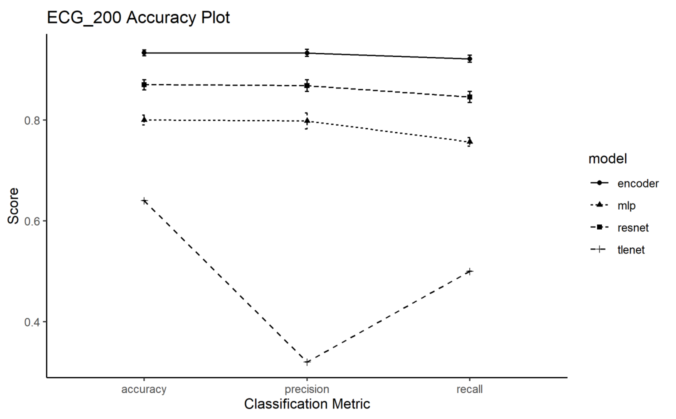


Fig. 10: Classification Metics (with average, sd error bars) for each model for *ECG200*

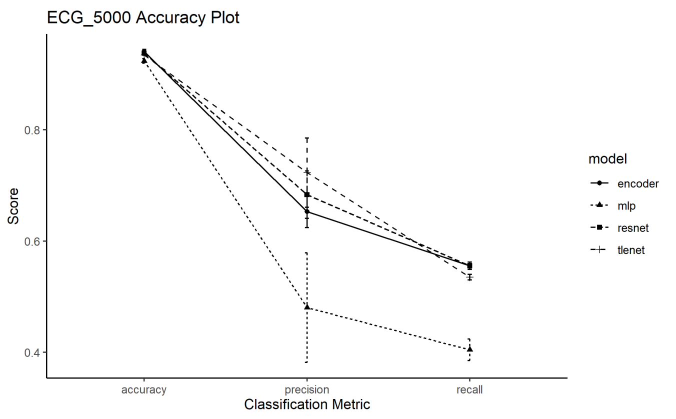


Fig. 11: Classification Metics (with average, sd error bars) for each model for *ECG5000*

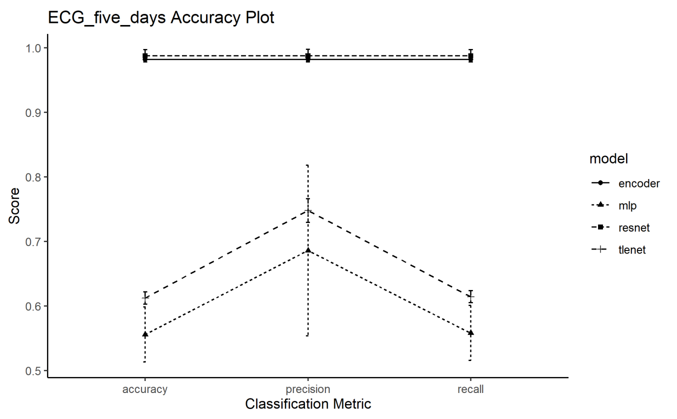


Fig. 12: Classification Metics (with average, sd error bars) for each model for *ECGfivedays*

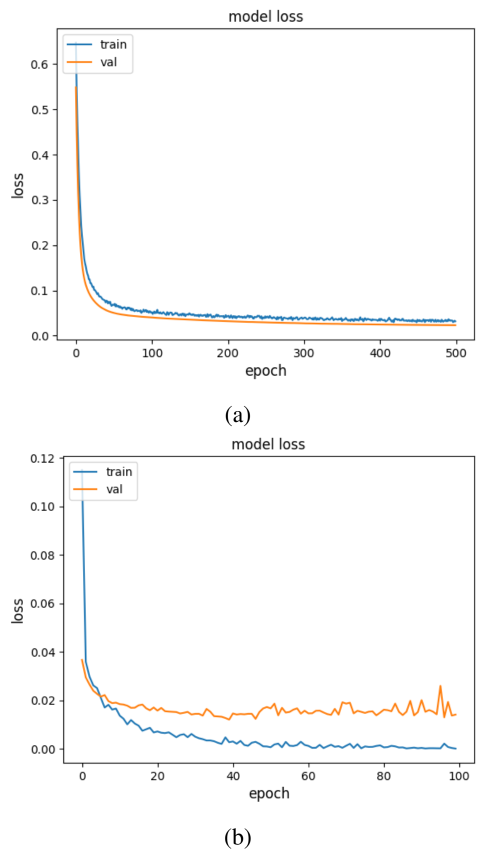
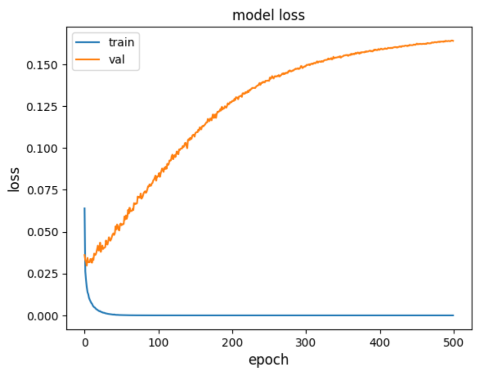
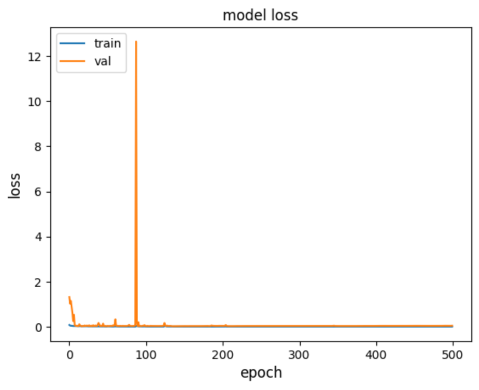


Fig. 13: Loss Metrics for (a) MLP and (b) Encoder for dataset *ECG1* across epochs

https://doi.org/10.1109/icdm.2017.10
[14] N. Mehdiyev, J. Lahann, A. Emrich, D. Enke, P. Fettke, and P. Loos, "Time series classification using deep learning for process planning: A case from the process industry," *Procedia Computer Science*, vol. 114, pp. 242–249, 2017. [Online]. Available: https://doi.org/10.1016/j.procs.2017.09.066
[15] P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. Shroff, "Timenet: Pre-trained deep recurrent neural network for time series classification," 2017.
[16] D. Rajan and J. J. Thiagarajan, "A generative modeling approach to limited channel ecg classification," 2018.
[17] W. Aswolinskiy, R. F. Reinhart, and J. Steil, "Time series classification in reservoir- and model-space," *Neural Processing Letters*, vol. 48, no. 2, pp. 789–809, Dec. 2017. [Online]. Available: https://doi.org/10.1007/s11063-017-9765-5
[18] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, "Reservoir computing approaches for representation and classification of multivariate time series," 2020.
[19] N. Chouikhi, B. Ammar, and A. M. Alimi, "Genesis of basic and multi-layer echo state network recurrent autoencoders for efficient data representations," 2018.
[20] Q. Ma, L. Shen, W. Chen, J. Wang, J. Wei, and Z. Yu, "Functional echo state network for time series classification," *Information Sciences*, vol. 373, pp. 1–20, Dec. 2016. [Online]. Available: https://doi.org/10.1016/j.ins.2016.08.081
[21] H. Chen, F. Tang, P. Tiño, A. Cohn, and X. Yao, "Model metric co-learning for time series classification." pp. 3387 – 3394, 2015. [Online]. Available: http://eprints.whiterose.ac.uk/92859/
[22] Z. Che, "Decade : A deep metric learning model for multivariate time series," 2017.
[23] Z. Gong, H. Chen, B. Yuan, and X. Yao, "Multiobjective learning in the model space for time series classification," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 918–932, Mar. 2019. [Online]. Available: https://doi.org/10.1109/tcyb.2018.2789422
[24] N. Papernot and P. D. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," *CoRR*, vol. abs/1803.04765, 2018. [Online]. Available: http://arxiv.org/abs/1803.04765
[25] *Efficient backprop*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2012, pp. 9–48.
[26] "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.
[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerat-



(a)



(b)

Fig. 14: Loss Metrics for (a) ResNet and (b) t-LeNet for dataset *ECG1* across epochs

ing deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[28] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *CoRR*, vol. abs/1607.08022, 2016. [Online]. Available: http://arxiv.org/abs/1607.08022

[29] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[30] C. Gallicchio and A. Micheli, "Deep echo state network (deepesn): A brief survey," *CoRR*, vol. abs/1712.04323, 2017. [Online]. Available: http://arxiv.org/abs/1712.04323

[31] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004. [Online]. Available: https://science.sciencemag.org/content/304/5667/78

[32] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The ucr time series archive," 2019.

[33] R. T. Olszewski, C. Faloutsos, and D. B. Dot, "Generalized feature extraction for structural pattern recognition in time-series data," 2001.

[34] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet," *Circulation*, vol. 101, no. 23, Jun. 2000. [Online]. Available: https://doi.org/10.1161/01.cir.101.23.e215

[35] Y. Chen, Y. Hao, T. Rakthanmanon, J. Zakaria, B. Hu, and E. Keogh, "A general framework for never-ending learning from time series streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1622–1664, Oct. 2014. [Online]. Available: https://doi.org/10.1007/s10618-014-0388-4

[36] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[38] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 392–401. [Online]. Available: https://doi.org/10.1145/2623330.2623613