Neural Architecture Search for Time Series Classification

Hojjat Rakhshani, Hassan Ismael Fawaz, Lhassane Idoumghar, Germain Forestier, Julien Lepagnot, Jonathan Weber, Mathieu Brevilliers, Pierre-Alain Muller

Neural Architecture Search for Time Series Classification

Hojjat Rakhshani, Hassan Ismail Fawaz, Lhassane Idoumghar, Germain Forestier,
Julien Lepagnot, Jonathan Weber, Mathieu Brévilliers, Pierre-Alain Muller
Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France – firstname.lastname@uha.fr

***Abstract*—Neural architecture search (NAS) has achieved great success in different computer vision tasks such as object detection and image recognition. Moreover, deep learning models have millions or billions of parameters and applying NAS methods when considering a small amount of data is not trivial. Unlike computer vision tasks, labeling time series data for supervised learning is a laborious and expensive task that often requires expertise. Therefore, this paper proposes a simple-yet-effective fine-tuning method based on repeated k-fold cross-validation in order to train deep residual networks using only a small amount of time series data. The main idea is that each model fitted during cross-validation will transfer its weights to the subsequent folds over the rounds. We conducted extensive experiments on 85 instances from the UCR archive for Time Series Classification (TSC) to investigate the performance of the proposed approach. The experimental results reveal that our proposed model called NAS-T reaches new state-of-the-art TSC accuracy, by designing a *single* classifier that is able to beat HIVE-COTE: an *ensemble* of 37 individual classifiers.**

***Index Terms*—Neural architecture search, times series classification, metaheuristics, deep learning**

I. Introduction

Nowadays, machine learning models have been popularly used in perceptual tasks by both academic and industrial researchers. These models are designed to find solutions for some specific Machine Learning (ML) tasks without being straightforward to apply an existing ML pipeline to a new domain and still have superior results [1]. Hence, machine learning experts have to construct a specialized ML pipeline for each given supervised learning task. The extra degree of freedom from the design space could make this process very time-consuming and has motivated a demand for Neural Architecture Search (NAS) methods that can be adopted easily without any expert knowledge [2], [3], [4], [5].

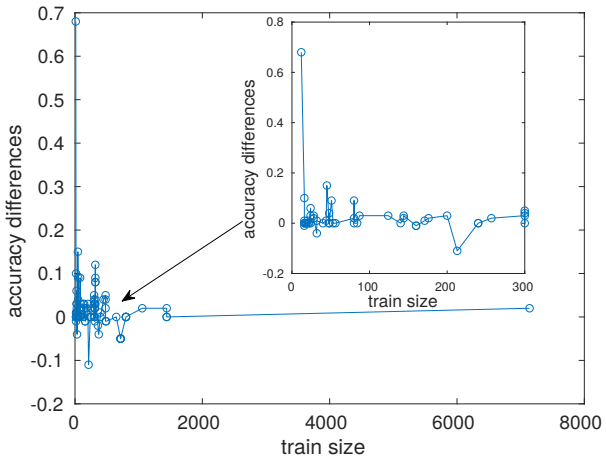
The existing methods usually employ random search [6], grid search [7], reinforcement learning [3], Bayesian optimization [8], evolutionary algorithms [9] and gradient-based methods [10] to explore the space of neural architectures. Although they give rise to a large number of studies for reporting more accurate classifiers, researchers are still faced with the challenge of computationally expensive simulations. This is primarily due to the huge number of parameters, often in the range of millions, which is associated with deep learning (e.g. [3] used 800 GPUs concurrently to generate a computer vision model for Cifar-10). Furthermore, neural networks mainly require extremely large quantities of data to be trained for a specific task which are unavailable in many real-world problems. Consequently, there has been a surge of interest in minimizing the design complexities of the NAS methods [11], [12], [13], [2], [14].

Time series classification (TSC) is one of those real-world problems revolving around small data sets, existing in cybersecurity [15], health monitoring data [16] and human mobility [17]. Traditionally, TSC tasks are tackled with non deep learning approaches such as Support Vector Machines and Nearest Neighbor classifiers [18]. Nevertheless, a recent significant amount of research efforts have been spent to embrace Convolutional Neural Networks (CNNs) for solving domain agnostic end-to-end TSC problems [19], [20], [21], [22], [23]. The aforementioned approaches, however, exclude NAS methods which can be a valuable tool to improve their empirical performance. Meanwhile, applying NAS methods on small TSC datasets often result in overfitting [24]; as presented in Figure 1. The results in the latter figure are obtained using a hyperparameter optimization of ResNet [22] on 85 data sets from the UCR archive [25]. We investigated the performance of the hyperparameter optimization on 85 different configuration TSC scenarios, which are defined by 11 control parameters. The details about optimized hyperparameters is presented in Table I. Our experimental procedure follows training, validation and test phases. The loss function for the model is categorical cross entropy. After finishing the automatic configuration, best configuration based on the validation accuracy is used to measure the performance of the optimized ResNet on the unseen test instances. The number of function evaluation is set to 100 and we repeat the experiment for 10 run using random search. As presented in the paper, the over-fitting was the main problem to continue with the aforementioned method.

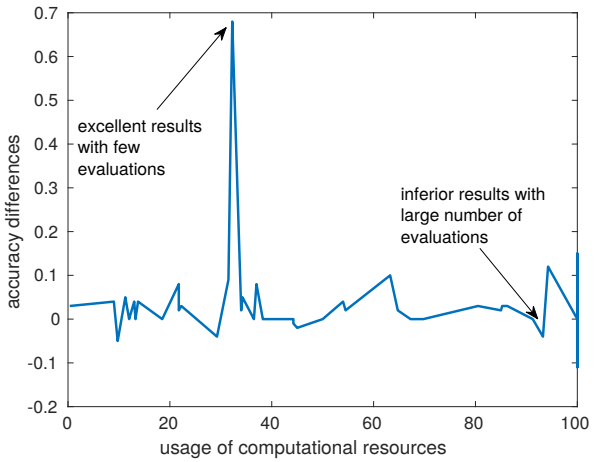
TABLE I: Details of the optimizing hyperparameters in ResNet for the TSC task.

Hyperparameter	Values	Default
Learning rate of Adam	{0.0001,0.001,0.01,0.1,0.2,0.3, 0.4,0.5,0.6,0.7,0.8,0.9}	adaptive
Batch size	{8,16,32,64,128,256}	adaptive
Features in 1th Layer	{8,16,32,64,128,256}	64
Features in 2th Layer	{8,16,32,64,128,256}	64
Features in 3th Layer	{8,16,32,64,128,256}	64
Features in 4th Layer	{8,16,32,64,128,256}	128
Features in 5th Layer	{8,16,32,64,128,256}	128
Features in 6th Layer	{8,16,32,64,128,256}	128
Features in 7th Layer	{8,16,32,64,128,256}	128
Features in 8th Layer	{8,16,32,64,128,256}	128
Features in 9th Layer	{8,16,32,64,128,256}	128

 DOI: https://doi.org/10.1109/IJCNN48605.2020.9206721



(a) Accuracy gain vs train size



(b) Accuracy gain vs computational budget

Fig. 1: Figure (a) shows the difference in accuracy with respect to the train size, while Figure (b) presents the trade-off between the computational resource usage and the gain in accuracy.

The current methods to tackle this problem rely on strategies such as fine-tuning [26]. In the same direction, this paper proposes a complementary solution, applicable when having different ML tasks to solve with small time series data sets. The main idea is to integrate the repeated k-fold cross-validation into the fine tuning process and transfer the learned weights across the folds instead of overwriting them. Thus, useful information already learned is shared across different folds which will be used to build the final classification model. We adopted an evolutionary algorithm to optimize the ResNet architecture. The process includes replacing the last few layers of a pre-trained network, with some randomly initialized ones. Then, the weights of the modified model are trained via backpropagation while freezing the weights of the pre-trained feature extraction network. This could be applied to all the layers of the model by using a smaller learning rate (to avoid losing previous knowledge), or possibly fine-tuning the weights of some higher-level portion of the model and treat the rest of the layers as fixed feature extractor components. We investigate how this policy may be used as an alternative to the traditional fine-tuning approach for TSC problem. The main contributions of this paper can be summarized as follows:

- We provide the first NAS-based for solving domain agnostic TSC problems.
- We reach new state-of-the-art results for TSC, when evaluating on the UCR archive.
- We show how a simple yet effective fine-tuning technique allows us to build a very accurate model.
- We provide the first open source framework for automatically building ResNet models for TSC.

II. Background

In this section, we will first provide some preliminary definitions, followed by a brief overview of deep learning approaches for TSC.

A. Time series classification problem

Assume that time series $X = [x_1, x_2, \ldots, x_T]$ is an ordered set of values $x_i \in \mathbb{R}$, with T denoting the length of the time series X. Also consider a dataset $D = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}$ to be a collection of pairs (X_i, Y_i) where X_i is a time series with Y_i as its corresponding label. TSC then consists of designing a classifier on D in order to map from the raw feature space X to a probability distribution over the set of unique classes in D. For many decades, the Nearest Neighbor (NN) classifier coupled with the Dynamic Time Warping (DTW) distance, has been the favorite classifier in many TSC tasks [18]. A recent empirical evaluation of TSC algorithms [27] showed that ensembling techniques such as the Hierarchical Vote Collective of Transformation-Based Ensembles (HIVE-COTE) significantly outperforms other individual classifiers such as NN-DTW. Although HIVE-COTE contained more than 37 classifiers, it is until recently that deep learning has been considered as a potential domain agnostic classifier of time series data, therefore, we will describe in the next subsection the recent achievements of neural networks for TSC problems.

B. Deep learning for time series classification

Deep learning has revolutionized the field of computer vision and is currently being adopted in many natural language processing tasks as well as reaching state-of-the-art performance in various speech recognition systems [28]. Inspired by this recent success of deep learning in these versatile fields, researchers started adopting these neural network algorithms for TSC.A group of authors [22] proposed the residual architecture (ResNet) to show how deep learning methods perform on time series data. In another work proposed by [29], a novel Fully CNN architecture was designed to reach state of the art performance for surgical skills evaluation from kinematic multivariate time series data. In [30], Autonomous Deep

Learning has been proposed for data stream problems. In [31], deep CNNs were used for human activity recognition from wearable sensors data. In summary, deep neural networks are being leveraged to improve current state-of-the-art performance in many TSC fields [20]. For domain agnostic TSC, ResNet is considered the state-of-the-art architecture when evaluated on the 85 datasets from the UCR archive [20]. However, we should emphasize that currently ResNet's architecture and its hyperparameters were not optimized yet for the underlying task, unlike HIVE-COTE whose hyperparameters were highly optimized [27]. The latter observation constitutes the main motivation of this paper: we believe ResNet could benefit from a hyperparameter optimization regime in order to reach state-of-the-art performance for TSC, similarly to how the NN-DTW benefited significantly from a cross-validation scheme to learn a specific Warping Window (WW) for each TSC dataset [18].

III. Residual neural networks

Deep Residual Networks (ResNets) were first proposed by [32] for image recognition tasks. Since its introduction in 2016, ResNet became one of the most adopted deep learning architectures in various domains such as object recognition [33], speech recognition [34] and many other natural language processing tasks [35]. In [22], a relatively deep residual network was proposed for classifying domain agnostic univariate time series data. More recently, ResNet is being used in many TSC domains such as in healthcare, where a deep ResNet was designed to diagnose with expert level accuracy irregular heart rhythms (arrhythmias) from single-lead electrocardiography signals [36]. Furthermore, ResNet showed a great performance when predicting urban building energy consumption [37], detecting road anomalies from smart-phones sensors data [38] and recognizing human activity from wearable sensors time series data [39]. Given the aforementioned success of ResNet for TSC, we decided to adopt the same architecture proposed in [22] for our fine-tuning optimization algorithm. The architecture is comprised of three main residual blocks, where each block contains three fully convolutional layers of respectively 8, 5 and 3 as kernel length each. The number of one-dimensional filters for the first block is fixed to 64, whereas for the second and third block this hyperparameter is set to 128. All convolutional layers employ a batch normalization operation to speed up the training process [40], with the Rectified Linear Unit (ReLU) as activation function [41]. Finally, we emphasize that for each residual block the input is fed with a linear shortcut to the output of the current block. This latter connection constitutes the main characteristic of a residual network which allows a direct flow of the gradient thus mitigating the vanishing gradient problem [32]. In conclusion, the approach described in this paper will consist of pre-training the already validated ResNet architecture, then using a meta-learning algorithm to add up to 15 hybrid layers making the final network's depth twice as deep as ResNet's. See Fig. 2 for an overview of our proposed framework.

IV. Methodology

This section presents a fine-tuning method to explore the possibility of automatically designing an enhanced ResNet network architecture for the TSC problem. We first propose a way of representing the network using linear strings with fixed length as genotype, and nonlinear entities as phenotype. The introduced NAS for TSC (NAS-T) method applies search operators directly on the genotype, while decodes a genotype into a phenotype only for evaluation purpose. The cost of each architecture is defined as its average loss validation error, which is obtained via fine-tuning the network using repeated k-fold cross-validation on the training set. The key point here is that all of the folds which NAS-T ends up iterating over will transfer their weights to the subsequent ones. The search algorithm then minimizes the cost function in order to deliver new architectures with strong empirical performances.

A. Design space

We provide a fixed length representation for an architecture which allows us to reduce the size of the search space as presented in Figure 3. This chain-structured method considers a limited number of layers which takes as inputs, the output tensors of the previous layers. For flexibility, NAS-T only allows data to flow in one direction: from a lower-numbered to a higher numbered layer. Note that even with such a relatively simple representation the total number of possible architectures will grow exponentially. Meanwhile, the complexity for building a graph structure is completely eliminated. We represent the network structure as a sequence of n computational layers. Each layer contains one computational node which is transformed from the previous layer. Given a sequence of layers, the output of the network is served as the concatenation of the computational nodes, i.e., $O = L_n \cdot L_{n-1} \cdot L_1$. Accordingly, the search space is parametrized by considering the type of the layer such as pooling or convolution; maximum number of layers; and the hyperparameters associated with each layer.

The NAS-T takes into account the highly popular computational modules as the candidate node functions set. These nodes are called Convolution, ZeroOp, MaxPooling, Dropout, Dense and Activation. All of these nodes consider one-dimensional tensor defined by the number of the input training data and dimension of TSC problem (i.e. the length T). If ZeroOp is selected, it means the block is skipped. For the convolutional layer, the kernel size is set to 3 followed by a batch normalization layer with a zero padding to maintain the length of the input time series throughout the convolutions. Thereafter, we have the Dense node which consists of a dense layer followed by rectified linear units and a dropout layer. The hyperparameters for the Convolution, Activation, Dropout and Dense layers are as follows: Number of filters = {8,16,32,64,128,256}, activation function = {'softmax', 'elu', 'selu', 'relu', 'sigmoid', 'linear'}, number of neurons = {4, 16, 32, 64, 128, 256}, dropout rate [0, 0.5]. Furthermore, we added the MaxPooling operation with a pooling size equal to 3 and a zero padding strategy. The final layer consists of a Dense fully-connected layer with softmax as the activation function

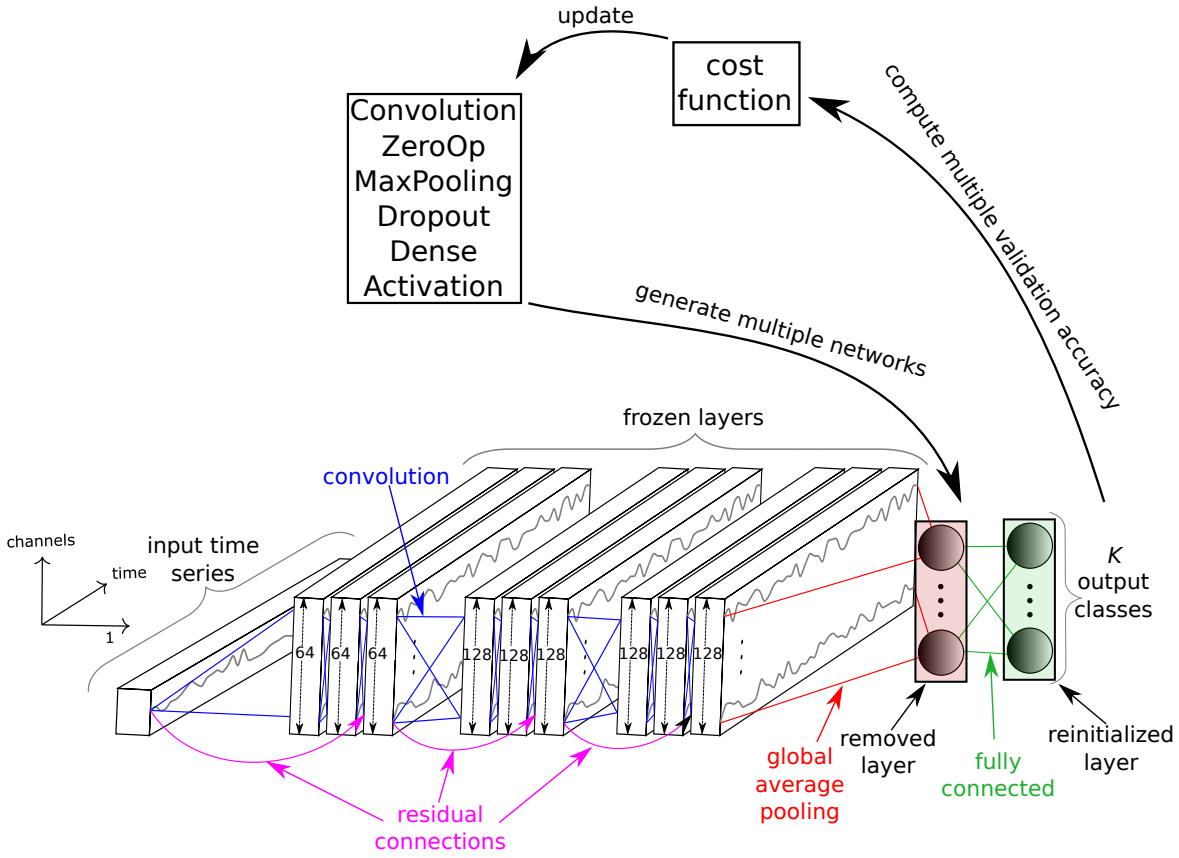


Fig. 2: Neural architecture search framework of deep residual networks for time series classification

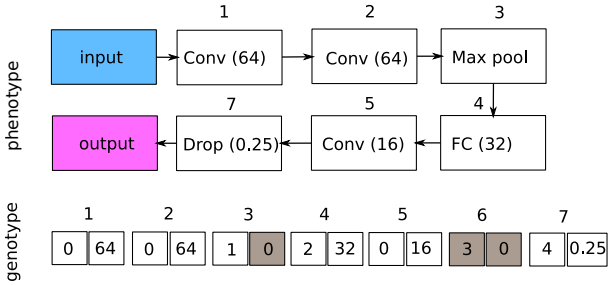


Fig. 3: An example of the adopted genotype and phenotype representation in NAS-T with $n = 7$. Here, some design parameters are conditional and are not expressed in the phenotype.

and a number of neurons equal to the number of classes in the dataset. So, the total number of possible architectures in the design space is equal to 6^{2n}.

In NAS-T, solutions are composed of a set of genotypes each with equal length which should be decoded into ML architectures (i.e., phenotype). The length of each solution is proportional to the maximum number of layers. For flexibility, it is possible that a specific part of each solution won't be used (e.g., due to the ZeroOp). We note that in the first round, we skip all the Dense layers. Figure 3 gives an example of decoding a genotype representation into a phenotype.

B. *Fine-tuning the ResNet*

After generating different architectures, NAS-T should measure their performance in order to maximize the expected validation accuracy of the models in the later iterations. We should note that holdout method is not recommended for model evaluation when small amount of data is available [42]. Consequently, the most common method for model evaluation, namely cross-validation, is adopted in NAS-T. The main goal is to testify the model's generalization ability by dividing the dataset into k different folds. In each round, $k-1$ folds are merged to form a training set and one fold is used for validation. The arithmetic mean over the validation accuracy of the k different fitted models will be used as the performance estimates.

NAS-T adopts a fine-tuning strategy to train these k-folds since it is computationally less expensive, given in Figure 4. Considering the ResNet architecture, we first train the standard ResNet network using the complete training set. Next, we remove the global average pooling and the final fully connected softmax layer. Once this has been done, the new generated architecture is attached to the model. Here, the weights of the newly added layers will be fine tuned using the Adam optimizer [43]. For the first fold, all the weights of the new layers are randomly initialized and the cost function will be minimized subject to the added weights. After that, the weights of the first trained fold will be used as a source of information

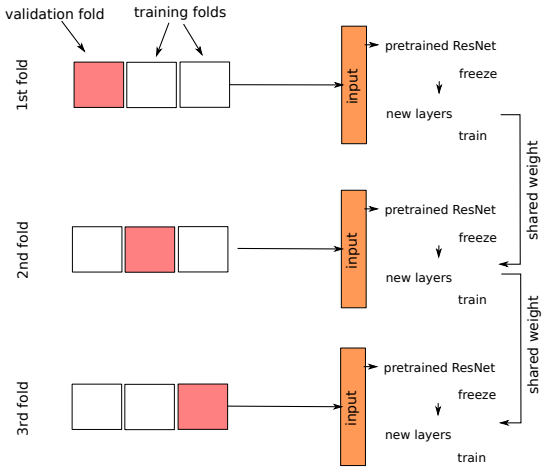


Fig. 4: Illustration of the adopted k-fold cross-validation with $k = 3$

to initialize the weights for the second fold, and so on.

C. Search method

The NAS-T uses the standard differential evolution (DE) [44] algorithm to generate and evolve a population of candidate architectures. DE finds an optimized architecture by iteratively improving the candidate solutions with regard to their average validation loss during the k-fold cross-validation. Different from traditional evolutionary algorithms, DE uses the scaled differences of vectors to produce new candidate solutions in the population. Hence, no separate probability distribution should be used to perturb the population members [44]. The DE is also characterized by the advantages of having few parameters and ease of implementation. Basically, it works through a particular sequence of steps. First, it creates an initial population that sampled uniformly at random within the search bounds. Thereafter, three components namely mutation, crossover and selection are adopted to evolve the initial population. The mutation and crossover are used to create new solutions, while selection determines the solutions that will breed a new generation. The algorithm remains inside a loop until stopping criteria are met. More precisely, DE starts with a randomly initialized population of parameter vectors the so-called individuals. Each such individual represents a $D = 2 \times n$ dimensional vector of fixed length genotypes. The *ith* individual of the population at generation G could be denoted as follows:

$$\overrightarrow{X}_{G,i} = [x_{G,i,1}, x_{G,i,2}, x_{G,i,3}, ..., x_{G,i,D}] \quad j = 1, 2, ..., D \tag{1}$$

For each individual, both upper and lower bounds of the decision variables should be restricted to their minimum and maximum values. Once the initialization search ranges have been determined, DE assigns (at $G = 0$) each individual a value from within the specified range as follows [44]:

TABLE II: Accuracy for the introduced fine-tuning approach with (NAS-T1) and without parameter sharing (NAS-T2) across the k-fold cross-validations.

Dataset	NAS-T1	NAS-T2
50words	**79.00**	76.04
Adiac	**83.00**	82.09
ArrowHead	85.00	**86.85**
Beef	83.00	**83.33**
BeetleFly	**95.00**	90.00
BirdChicken	**100**	95.00
Car	95.00	**96.23**
CBF	**99.33**	99.22
ChlorineConcentration	**87.13**	86.71
CinC_ECG_torso	**99.71**	98.04

$$x_{0,i,j} = min_j - r.(max_j - min_j) \quad j = 1, 2, ..., D \tag{2}$$

Where $r \in [0, 1]$ represents a uniformly distributed random number and *NP* denotes the population size. After initialization, the mutation operator produces new solutions by forming a mutant vector (trial vector) with respect to each parent individual (target vector). For each target vector, its corresponding trial vector can be generated by different mutation strategies. Each strategy employs different approaches to make a balance between the exploration and exploitation tendencies. We use the following strategy to do so:

$$\overrightarrow{V}_{G,i} = \overrightarrow{X}_{G,r1} + F.(\overrightarrow{X}_{G,r2} - \overrightarrow{X}_{G,r3}) \tag{3}$$

Here *r1, r2* $\in NP$ are different randomly generated integer numbers. Furthermore, *F* is a scaling factor $\in [0, 2]$ affecting the difference vector and *best* $\in NP$ is index of the best individual vector at generation G. Thereafter, DE applies a discrete crossover approach to each pair of the parent vector and its corresponding trial vector. The basic version of DE incorporates the binomial crossover as defined in [44]. Finally, it adopts a selection mechanism to choose the best individuals according to their validation loss for producing the next generation of population. DE compares performance of the trial and target architectures and copies the better one into next generation.

V. Experiments

A. Experimental setup

We evaluate our approach on the publicly available UCR time series data mining archive [25] with the corresponding results in TABLE II. The latter benchmark is comprised of 85 datasets from various domains such as ECG, electric consumption and food spectrography data, and has been considered a standard evaluation of TSC algorithms [27]. The number of training instances ranges between 16 and 8000 with a length that achieves a maximum of 2700 and a minimum of 24. We refer the interested reader to a thorough description of these datasets in [27]. We trained our models by leveraging the

parallel computation on a cluster of more than 60 Nvidia GPUs using the Keras and Tensorflow APIs[1].

We did a comparative experiment on the obtained results with and without weight sharing. For this ablation study, the experiments are computationally expensive and so they are reported only for first 10 alphabetically ordered datasets. In DE, population size is 100 and F is set to 0.5 in all experiments. The value of crossover rate, which controls the change of diversity in population, is chosen to be 0.9 [45]. For each dataset, the stopping criteria is when the NAS-T reaches 5000 evaluations, or 48 hours of computations on a single GPU. We investigated the performance of the NAS-T on 85 different configuration of TSC scenarios. The loss function for the model is the categorical cross-entropy.

After finishing the automatic configuration, the best configuration based on the validation accuracy is used to measure the performance of the optimized ResNet on the unseen test instances. As it can be noticed from the literature [20], the ResNet model shows a superior performance for the TSC task but with a high standard deviation. To this fact, we repeat the experiments using each configuration for 10 times in order to optimize the mean performance of the model. The obtained results are compared with stat-of-the-art methods including: HIVE-COTE, COTE, PF, ST, BOSS, EE and NN-DTW, which are further detailed in the next subsection. It is worth mentioning that HIVE-COTE is an ensemble model which combines the weighted votes of 37 different classifiers.

B. Results

Since NAS-T is the first proposed AutoML approach for domain agnostic TSC, we have compared the DE optimization algorithm to the random search. NAS-T achieved a Win/Tie/Loss record of 72/6/7 against a ResNet tuned with random search; as shown in Figure 5.

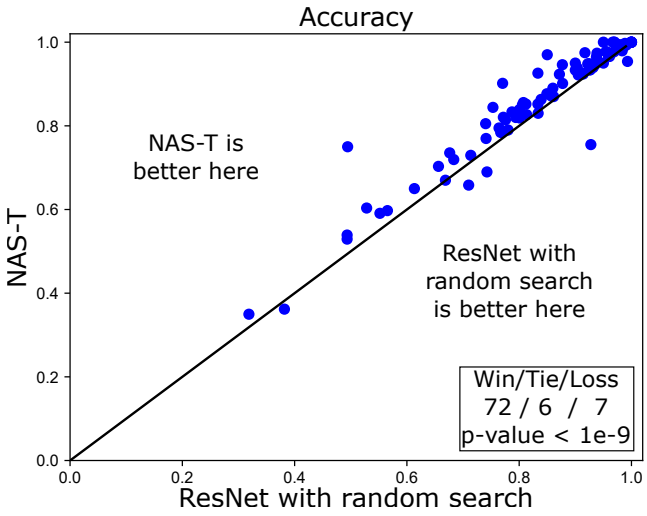


Fig. 5: Accuracy plots showing how NAS-T is setting new results against the random search

[1]https://github.com/ML-MHs/IJCNN2020

In order to compare multiple classifiers over several datasets, we adopted the Friedman test to first reject the null hypothesis as suggested by [46]. Then following the recommendations in [47], we performed a post-hoc analysis using the Wilcoxon signed-rank test with Holm's alpha correction with $\alpha = 0.05$ as initial value. For visualization, we used the critical difference diagram as proposed in [46], with a thick horizontal line showing a group of classifiers that are not significantly different. For example, Figure 6 shows the average rank comparison, with our proposed NAS-T approach achieving the highest performance over the 85 datasets from the UCR archive.

Note that the results for the NN-DTW-WW [27], Elastic Ensemble (EE) [48], Bag-Of-SFA-Symbols (BOSS) [49] and Shapelet Transform (ST) [50] were taken from the recent review for time series classification [27]. For Proximity Forest (PF) [51] and HIVE-COTE [52], the results were taken from their corresponding papers. Finally, note that over the last couple of years, most approaches for TSC were focusing on ensembling different classifiers [53], which is indeed the case for EE, BOSS, PF, ST and HIVE-COTE. Whereas with our optimization technique we are able to reach better performance with a single well designed deep neural network classifier.

Figure 7a depicts the accuracy plot comparing the original implementation of ResNet against NAS-T. We can clearly see the superiority and the benefit of optimizing the architecture with 72 wins out of 85 datasets and a p-value $<$ 1e-10. In Figure 7a, the "*DiatomSizeReduction*" dataset is a concrete example where the original ResNet implementation [22] performs extremely poorly (33% accuracy) than our optimized ResNet architecture (97% accuracy). In fact, it turned out that choosing a sub-optimal batch-size for "*DiatomSizeReduction*" would deteriorate extremely the results, which was the case with the original implementation of ResNet [22]. However, NAS-T was able to fine-tune the batch-size in order to maximize the accuracy of the model. The latter example is evident that choosing a network's hyperparameters is key to achieving superior results with deep learning. Furthermore, when comparing to the ensemble HIVE-COTE in Figure 7b, NAS-T is better on 49 out of 85 datasets with a p-value $<$ 0.08 when performing the Wilcoxon Signed-rank test.

Our results show how a single well designed and optimized neural network architecture is able to outperform HIVE-COTE, which is an ensemble of 37 different well optimized classifiers [52]. These results should convince the TSC community that single end-to-end deep learning classifiers are able to reach state-of-the-art results for TSC (currently achieved only using ensembling techniques), especially when adopting a meta-learning approach to design the neural network architecture.

VI. Conclusion

ResNet model has been recently explored for domain agnostic TSC problems. In the same direction, we proposed a fine-tuning method to enhance the classifier's imperial performance for the underlying TSC task. The introduced algorithm is based on two main components in order to offer the advantageous and accurate solutions. The first one is a

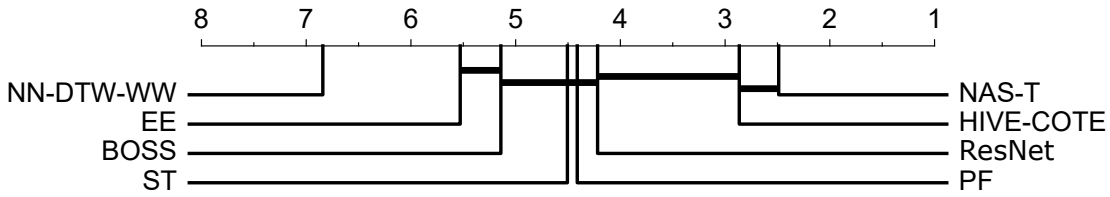
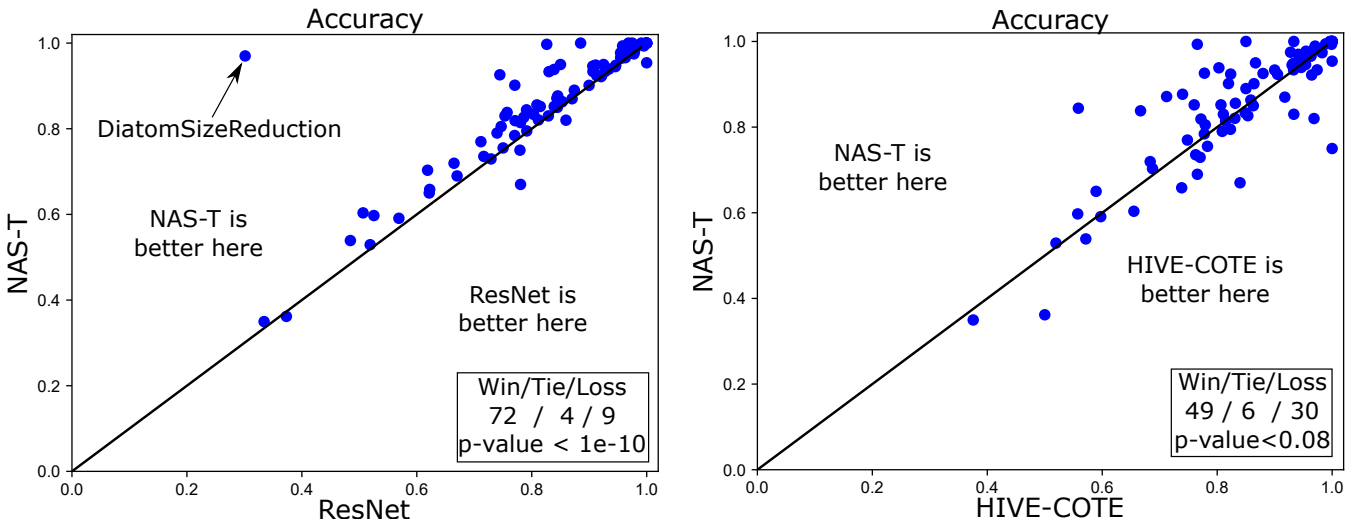


Fig. 6: Critical difference diagram showing the new state-of-the-art with our NAS-T approach.



(a) NAS-T vs the original ResNet

(b) NAS-T vs the ensemble HIVE-COTE

Fig. 7: Accuracy plots showing how our classifier NAS-T is setting new state-of-the-art results for TSC.

weight sharing strategy which plays a key role in reducing the computational cost during the training process. The second component is an evolutionary algorithm which is integrated to find more promising solutions by virtue of its exploration capability. The experiments show that the obtained per-dataset classifier outperforms the hand engineered ResNet model. We believe that NAS-T will help non-expert users to more effectively apply CNN models to their TSC applications. In the future, we would like to explore designing neural networks from scratch for classifying time series data with limited resources by minimizing the model's complexity.

Acknowledgments

The authors would like to thank the providers of the UCR datasets as well as Nvidia Corporation for the GPU Grant and the Mésocentre of Strasbourg for providing access to the GPU cluster.

References

[1] M.-A. Zöller and M. F. Huber, "Survey on automated machine learning," *ArXiv*, 2019.

[2] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *ArXiv*, 2018.

[3] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[4] F. Qi, Z. Xia, G. Tang, H. Yang, Y. Song, G. Qian, X. An, C. Lin, and G. Shi, "Darwinml: A graph-based evolutionary algorithm for automated machine learning," *ArXiv*, 2018.

[5] H. Rakhshani, L. Idoumghar, J. Lepagnot, and M. Brévilliers, "Mac: Many-objective automatic algorithm configuration," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2019, pp. 241–253.

[6] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *ArXiv*, 2016.

[7] J. Y. Hesterman, L. Caucci, M. A. Kupinski, H. H. Barrett, and L. R. Furenlid, "Maximum-likelihood estimation with a contracting-grid search algorithm," *IEEE transactions on nuclear science*, vol. 57, no. 3, pp. 1077–1084, 2010.

[8] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*, 2018.

[9] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *IEEE Congress on Evolutionary Computation*, 2018, pp. 1–8.

[10] N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "Gradient based evolution to optimize the structure of convolutional neural networks," in *IEEE International Conference on Image Processing*, 2018, pp. 3438–3442.

[11] M. Feurer, J. T. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[12] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 19–34.

[13] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[15] M. Rahman, M. Rahman, B. Carbunar, and D. H. Chau, "Fairplay: Fraud

and malware detection in google play," in *SIAM International Conference on Data Mining*, 2016, pp. 99–107.
[16] T. Ma, C. Xiao, and F. Wang, "Health-ATM: A deep architecture for multifaceted patient health record representation and risk prediction," in *SIAM International Conference on Data Mining*, 2018, pp. 261–269.
[17] H. Shi, H. Cao, X. Zhou, Y. Li, C. Zhang, V. Kostakos, F. Sun, and F. Meng, "Semantics-aware hidden markov model for human mobility," in *SIAM International Conference on Data Mining*, 2019, pp. 774–782.
[18] C. W. Tan, M. Herrmann, G. Forestier, G. I. Webb, and F. Petitjean, "Efficient search of the best warping window for dynamic time warping," in *SIAM International Conference on Data Mining*, 2018, pp. 225–233.
[19] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," *ArXiv*, 2016.
[20] 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*, 2019.
[21] A. Borovykh, S. Bohte, and K. Oosterlee, "Conditional time series forecasting with convolutional neural networks," in *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence*, 2017, pp. 729–730.
[22] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Neural Networks, 2017 International Joint Conference on*. IEEE, 2017, pp. 1578–1585.
[23] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Adversarial attacks on deep neural networks for time series classification," in *IEEE International Joint Conference on Neural Networks*, 2019.
[24] ——, "Data augmentation using synthetic data for time series classification with deep residual networks," in *International Workshop on Advanced Analytics and Learning on Temporal Data, ECML PKDD*, 2018.
[25] 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," *ArXiv*, 2018.
[26] A. Anderson, K. Shaffer, A. Yankov, C. D. Corley, and N. O. Hodas, "Beyond fine tuning: A modular approach to learning on small data," *ArXiv*, 2016.
[27] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.
[28] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.
[29] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Accurate and interpretable evaluation of surgical skills from kinematic data using fully convolutional neural networks," *International Journal of Computer Assisted Radiology and Surgery*, pp. 1–7, 2019.
[30] A. Ashfahani and M. Pratama, "Autonomous deep learning: Continual learning approach for dynamic environments," in *SIAM International Conference on Data Mining*, 2019, pp. 666–674.
[31] R. Xi, M. Hou, M. Fu, H. Qu, and D. Liu, "Deep dilated convolution on multimodality time series for human activity recognition," in *International Joint Conference on Neural Networks*, 2018, pp. 1–8.
[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[38] F. S. Cabral, M. Pinto, F. A. L. N. Mouzinho, H. Fukai, and S. Tamura, "An automatic survey system for paved and unpaved road classification and road anomaly detection using smartphone sensor," in *IEEE International Conference on Service Operations and Logistics, and Informatics*, 2018, pp. 65–70.
[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, vol. 4, 2017, p. 12.
[34] Y. Zhang, W. Chan, and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017, pp. 4845–4849.
[35] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for natural language processing," *ArXiv*, 2016.
[36] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, G. H. Tison, C. Bourn, M. P. Turakhia, and A. Y. Ng, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature medicine*, vol. 25, no. 1, p. 65, 2019.
[37] R. Aras, A. Nutkiewicz, and M. O'Krepki, "A neural network approach to predicting urban building energy consumption," 2018.
[39] H. Abdelkawy, N. Ayari, A. Chibani, Y. Amirat, and F. Attal, "Deep HMResNet model for human activity-aware robotic systems," in *Artificial Intelligence for Human-Robot Interaction Symposium at AAAI-FSS*, 2018.
[40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.
[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
[42] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *ArXiv*, 2018.
[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[44] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.
[45] H. Rakhshani and A. Rahati, "Snap-drift cuckoo search: A novel cuckoo search optimization algorithm," *Applied Soft Computing*, vol. 52, pp. 771 – 794, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494616305075
[46] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Machine Learning Research*, vol. 7, pp. 1–30, 2006.
[47] A. Benavoli, G. Corani, and F. Mangili, "Should we really use post-hoc tests based on mean-ranks?" *Machine Learning Research*, vol. 17, no. 1, pp. 152–161, 2016.
[48] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 565–592, 2015.
[49] P. Schäfer, "The boss is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.
[50] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *ACM SIGKDD International Conference on Knowledge discovery and data mining*, 2012, pp. 289–297.
[51] B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, "Proximity forest: an effective and scalable distance-based classifier for time series," *Data Mining and Knowledge Discovery*, vol. 33, no. 3, pp. 607–635, 2019.
[52] J. Lines, S. Taylor, and A. Bagnall, "Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 5, pp. 52:1–52:35, 2018.
[53] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep neural network ensembles for time series classification," in *IEEE International Joint Conference on Neural Networks*, 2019.