A novel surrogate model for variable-length encoding and its application in optimising deep learning architecture.

DANG, T., NGUYEN, T.T., MCCALL, J., HAN, K. and LIEW, A.W.-C.

2024

A novel surrogate model for variable-length encoding and its application in optimising deep learning architecture

Truong Dang, Tien Thanh Nguyen, John McCall
National Subsea Centre, Robert Gordon University
Aberdeen, UK

Kate Han
Salford Business School, University of Salford
Manchester, UK

Alan Wee-Chung Liew
School of ICT, Griffith University
Queensland, Australia

***Abstract*—Deep neural networks (DNN) has achieved great successes across multiple domains. In recent years, a number of approaches have emerged on automatically finding the optimal DNN configurations. A technique among these approaches which show great promise is Evolutionary Algorithms (EA), which are based on observations from natural, biological processes. However, since the EA needs to evaluate multiple DNN candidates, and if the training time for a DNN is large, then the required time would be very large. A potential solution is to use Surrogate Assisted Evolutionary Algorithm (SAEA), in which a surrogate model is used to predict performance of DNNs without training. It is noted that all popular surrogate models in the literature require a fixed-length input, while encodings of a DNN are usually variable-length, since a DNN structure is very complex and its depths, sizes, etc. cannot be known beforehand. In this paper, we propose a novel surrogate model for variable-length encoding to optimise deep learning architecture. An encoder-decoder model is used to convert the variable-length encoding into a fixed-length representation, which is used as inputs to the surrogate model to predict the DNN performance without training. The weights of the encoder-decoder model are found via training on the variable-length data, with the targets being the same as the inputs, while the surrogate model is trained on the encoder output in the encoder-decoder model. In this study, a Long Short-Term Memory (LSTM) model is used as the encoder and decoder. Our proposed variable-length encoding based surrogate model is tested on a well-known method which evolves optimal Convolutional Neural Networks (CNNs). The experimental results show that our proposed method has competitive performance while significantly reducing the time of optimisation process.**

***Index Terms*—surrogate model, variable-length encoding, deep learning, encoder-decoder, sequence-to-sequence**

I. Introduction

In recent years, deep learning has brought many great breakthroughs in multiple areas, such as computer vision and natural language processing. Deep-learning methods are based on composing simple but non-linear modules which transform the representation at one level (starting with the raw input) into a representation at a higher level, thereby allowing very complex functions to be learned [1]. However, the configurations of deep learning models have been normally found manually via trial-and-errors. This is a very time-consuming and laborious process, due to the fact that there is no clear guidance on how to create the best deep learning model, researchers have to perform a large number of experiments to find the best configurations. In recent years, there have been many efforts on automatically finding the best DNN configurations using EA [2], reinforcement learning [3], Bayesian optimisation [4] and gradient-descent based approach [5]. Among these approaches, EA show great promise, due to their capabilities to work on large search space, faster convergence compared to other methods, diverse representational abilities, and its multi-objective extension has been shown to have been successfully applied to solve problems with multiple objective functions [6].

A significant challenge in employing EA for deep learning lies in the considerable amount of time required for experimentation. An example is [7] which required 4 days to evolve a CNN on the FashionMNIST dataset [8] using 2 GPUs. This is due to the fact that the EA run through a number of generations, in which each candidate's fitness is evaluated and then a number of operators are applied on the candidates. In the case of deep learning, the fitness value of a candidate is usually calculated based on the performance of this candidate on a validation set. If the training time is large, it would require a large amount of time to run. A potential approach to alleviate this difficulty is by using Surrogate Assisted Evolutionary Algorithm (SAEA), in which a *surrogate model* is used to approximate the DNN performance without actually training it. Common choices of surrogate models include Radial Basis Function, Support Vector Machine [9], Random Forest [10] and XgBoost [11].

On the other hand, it should be noted that a DNN encoding for EA needs to be representative of all the DNN structures, such as the type of each layer (convolutional, pooling, fully-connected, etc.), number of filters for each layer, etc. Furthermore, the optimal depth of a DNN is not known in advance so that the DNN encoding must be of variable-length. However, the surrogate model can only work with a fixed-length input. This leads to the problem of finding a suitable method to make

the variable-length representation work with surrogate models.

In this paper, we propose a novel surrogate model for variable-length encoding to optimise deep learning architectures. Our proposed method is inspired by the successes of the sequence-to-sequence models [12]. It is known that sequence-to-sequence models receive a variable-length input, such as a series of text vectors, and process it into a fixed-length vector (encoder), which is then processed into an output which is of variable-length as well (decoder). We apply this idea for evolving DNNs in the framework of SAEA. The encoder is used to convert the variable-length encoding into a fixed-length representation, which allows the surrogate model to predict the DNN performance without training. The weights of the encoder-decoder model are found by training on the variable-length data, and once the training is complete the decoder is discarded. Although several choices for the encoder and decoder are available, in this study we choose Long Short-Term Memory (LSTM) [13] for this task because of its ability to compute a function of variable-length inputs [14], and its popularity and high performance in sequence-to-sequence tasks [12]. We test our proposed LSTM-based variable-length encoding surrogate model on a method namely EvoCNN [7] which evolves optimal CNNs. Our contributions are as follows:

- We propose a novel LSTM-based variable-length encoding surrogate model for optimising deep learning architecture. The LSTM-based encoder-decoder model is used to convert the variable-length encoding into a fixed-length representation, which is then used by the surrogate model to predict the DNN performance without training.
- We test our proposed model on a well-known neuroevolution method, and the results showed that our proposed model has competitive performance with the original evolution-based model without surrogate, while reducing the computational time by several orders of magnitude.

The paper is organized as follows. In Section 2, a brief review of the related works is provided. Our proposed model is introduced in Section 3. The details of experimental studies on several medical segmentation datasets are described in Section 4. Finally, the conclusion is given in Section 5.

II. Background and related works

A. Deep learning and optimizing deep learning architectures

Although DL can be powerful for solving various real-world classification and prediction problems, its complex nature in architectural design has always been a great challenge. Promising results of deep learning networks rely on optimal architectural design and parameter settings. In recent years, new approaches have emerged which seek to automatically find the best DNN configurations. Among these approaches, EAs have been attracting increasing attention in DNN automated optimization due to its ability to solve non-continuous, non-differentiable problems. An optimized hybrid deep learning DenseNet121 architecture was proposed in [15] for chest X-ray images for COVID-19 diagnosis. In this paper, the authors adopted the gravitational search optimization (GSA) optimization algorithm [16] for hyperparameters tuning of the DenseNet121 architecture. In [17], the authors proposed NEAT (NeuroEvolution of Augmenting Topologies) which explores and evolves various neural network structures automatically. NEAT begins with a minimal encoding and during evolution, new connections and nodes are added, and each gene in an encoding has a global innovation number for crossover. Miikkulainen et al. proposed CoDeepNEAT [18] for evolving DNNs based on the NEAT algorithm. In CoDeepNEAT, two populations, blueprints and modules (for overall DNN structure and for each component respectively), are evolved and for each candidate, each blueprint node is replaced with a randomly chosen chromosome module and the fitness of each blueprint and module is the average of the fitness of all DNN chromosomes having that blueprint or module. In [19], the authors proposed to evolve DNNs based on Inception-based module called cells, with each being connected to the previous two cells. Each cell is either a normal cell, which preserves the image size, or reduction size, which has a pooling operator of stride 2. The cells are constructed by applying five different pairwise combinations of the hidden states, and afterwards, the remaining hidden states are concatenated to give the final results.

B. Surrogate models

In EAs, several operations such as crossover and mutation are applied to the candidate population for some generations until the optimal solution is achieved. For each evolved candidate, it is necessary to evaluate its quality by using a fitness function. However, for many real-world optimization problems, significant computation time is required for fitness evaluation [9] which makes the use of EAs infeasible in these cases. In recent years, a new approach, called Surrogate Assisted Evolutionary Algorithms (SAEA) has been proposed for solving computationally expensive problems assisted by surrogate models and has achieved good performance in a limited computational budget [20]. The surrogate model can be any type of machine learning model, such as Random Forest [10] and XgBoost [11].

SAEAs can be divided into three broad categories: global, local, and hybrid methods. In global SAEAs, the surrogate model approximates the entire landscape of a given problem to improve the search capabilities of the EA [21]. An example is [22], in which the authors built a surrogate for the objective and constraint functions, and the surrogate functions were used to identify the feasible trial offspring with either the best predicted objective values or with the minimum number of predicted constraint violations. Luo et al. [23] used a one-layer feed-forward network as a surrogate model, combined with an encoder-decoder network for gradient-based DNN optimisation. The surrogate model and the encoder-decoder are trained jointly in a multi-task setting. However, for complex problems, a global surrogate model might not be able to accurately model the entire search space and prevent the search from reaching the optimal solution [24]. In local SAEAs, local

surrogate models, trained with the specific data located at a sub-space of the decision space, are used to assist EAs. For example, Yu et al. [25] adopted a local RBF surrogate model as an approximation landscape of the optimisation problem to assist social learning PSO (SL-PSO). A restart strategy was utilized to select top-ranked individuals to form the SL-PSO population. In [24], the authors proposed a memetic algorithm with simultaneous local searches using ensemble and smoothing surrogate models, to predict fitness values reliably and improve search results. On the other hand, hybrid methods seek to combine both global and local surrogate models. Liao et al. [26] considered the global and local modeling processes as two related tasks and use a multitask EA to solve them collaboratively while updating the global and local surrogate models based on the obtained solution. In [27], the authors proposed a feedback mechanism-driven SAEA where global RBFs were used to pre-screen candidate trials and local RBFs were employed to accelerate the local search.

C. Variable-length encoding

In evolutionary deep learning research, one of the crucial steps is to represent the deep learning configuration before sending it to optimisation algorithms. With respect to representative dimension design, the encoding of network architecture can be divided into fixed-length encoding and variable-length encoding. Even though variable-length encoding require specialized genetic operators, it is more adaptive and does not require human expertise such as optimal depth in advance [2]. In [7], the authors proposed EvoCNN, a variable-length encoding to evolve DNNs for image classification. EvoCNN represents a CNN as a variable-length encoding of convolutional, pooling and fully-connected layers, and customized crossover and mutation operators are developed for the evolution process. The architecture and weights are jointly evolved by including the mean and standard deviation of the weight initialization process in the encoding. A variable-architecture encoding strategy was proposed in [28] to realize an adaptive scalable DNN architecture search. The authors used a block-based encoding to simplify the design and search process, and a reinforcement operator is designed to help select the random evolution operators with a learning process. Sun et al. [29] proposed a variable-length encoding to optimise the network architecture from both the micro and macro levels at the same time. In this paper, there are two types of network blocks, pooling block and convolutions block. The encoding strategy in this paper depends on the number of blocks selected to construct the network as well as the configuration for each block. Specially designed operators demonstrate in the paper in detail the feasibility of the network after decoding. In [30], the authors proposed a variable-length encoding-based genetic algorithm to search for an ensemble learning optimal configuration. They designed chunk-based crossover and point-based mutation genetic operators for their proposed variable-length encoding.

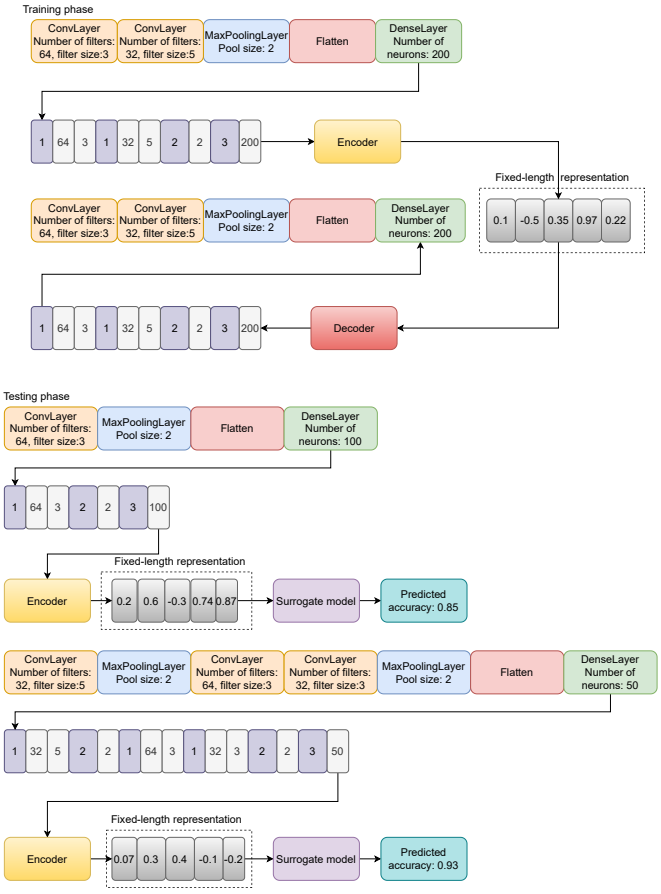

Fig. 1. An example of encoder and decoder in the proposed method

III. Proposed Method

The main ideas of our proposed method is illustrated in Figure 1. The training phase is shown in the upper part, while the testing phase is shown in the lower part. During the training phase, a given DNN architecture will be converted into a variable-length encoding. An example is given, in which the violet boxes show the layer type (1 denotes convolutional layer, 2 denotes max-pooling layer, and 3 denotes a dense layer). In the lower part, two example DNNs are shown with different number of layers, and the resulting encoding has different sizes as well. During the training phase, the variable-length DNN encoding will be used as input to an encoder, which converts the variable-length encoding into a fixed-length representation. Afterwards, a decoder will be used to convert the fixed-length representation to a variable-length output such that the output will be as close to the input as possible. This process, which is similar to that of an autoencoder, allows the weights of the encoder and decoder to be trained via an optimisation algorithm, such as the Adam algorithm.

In the testing phase, shown in the lower part of Figure 1, each DNN architecture is converted into a variable-length encoding, and sent to the encoder to create a fixed-length representation. However, unlike the training phase, the decoder

is not used but instead a surrogate model would use the fixed-length representation to predict the DNN performance without actually training the DNN. Two examples in the testing phase are shown, the first one is a four-layer DNN which is encoded as a vector of length 7, while the second example is a seven-layer DNN encoded as a vector of length 15. Both are converted to a fixed-length vector of length 5 via the encoder, then the surrogate model is used to predict the DNN performance, with the first one having a predicted accuracy of 0.85 while the surrogate model predicts that the second DNN would have an accuracy of 0.93.

Our work is loosely related to the work by Luo et al. [23] which also used an encoder-decoder architecture to optimise DNNs, however the authors of that work used a gradient-based approach for DNN optimisation. The authors also did not investigate the use of LSTM-based encoder-decoder to convert a variable-length DNN encoding into a fixed-length representation to be used by a surrogate model in the context of SAEA. Another related work is by Gong et al. [31] which also used a sequence-to-sequence based procedure to convert a variable-length encoding to a fixed-length representation. However, the authors did not investigate the use of surrogate models, and performed evolution on the fixed-length vector, while the newly generated candidates are converted back to the variable-length encoding using the decoder. In contrast, in our work, the evolutionary process is performed on the original variable-length space while the fixed-length representation is only used as the input for the surrogate model to predict the DNN performance without training.

Next, we describe how the main ideas of our proposed method is applied in the SAEA framework for optimising deep learning architectures. Let N_{cand} be the number of candidates, N_{gen} be the number of generations, in which each candidate denotes a different DNN architecture. The algorithm will be divided into two stages. During the first stage (cold start), the DNN candidates are trained normally for $N_{coldstart}$ generations, and evaluated on a validation set ($N_{coldstart} < N_{gen}$). The performance of each candidate are stored in a dataset $\mathbf{D}_{coldstart} = \{x_n^{var}, y_n\}_{n=1}^{L}$, where x_n^{var} denotes the variable-length DNN architecture encoding of the n^{th} candidate during the cold start stage, and y_n denotes its performance on the validation set after being trained, and $L = N_{coldstart} * N_{cand}$. Then, the encoder-decoder model will be used to train on $\{x_n^{var}\}$ such that the output of the decoder matches the input of the encoder as closely as possible (note that y_n is not used at this point). In other words, we solve the optimisation problem:

$$\min_{W_{Enc}, W_{Dec}} \frac{1}{L} \sum_{n=1}^{L} ||Dec(Enc(x_n^{var})) - x_n^{var}|| \qquad (1)$$

where Enc and Dec denotes the encoder and decoder, and W_{Enc} and W_{Dec} denotes their respective weights. Once the encoder-decoder model has been trained, we discard the decoder and use the encoder to convert the variable-length inputs into a fixed-length representation. More specifically, we create the surrogate data $\mathbf{D}_{surr} = \{x_n^{fixed}, y_n\}_{n=1}^{L}$, where $x_n^{fixed} = Enc(x_n^{var})$ is the fixed-length representation of x_n^{var}. A surrogate model $G(.)$ is then used to learn the relationships between x_n^{fixed} and y_n by training on $\mathbf{D}_{surr}$ to solve the optimisation problem:

$$\min_{W_G} \frac{1}{||\mathbf{D}_{surr}||} \sum_{n=1}^{||\mathbf{D}_{surr}||} ||G(Enc(x_n^{var})) - y_n|| \qquad (2)$$

where W_G is the parameters of the surrogate model, and $||\mathbf{D}_{surr}||$ is the size of the surrogate data (which will be updated during the training process), and $G(Enc(x_n^{var}))$ is the predicted performance by the surrogate model G of the DNN architecture encoded by x_n^{var}. Note that the surrogate model G can be a traditional machine learning model, such as Random Forest or XgBoost, or a deep learning model, such as CNN.

Another problem is the choice of the encoder and decoder, since the encoder must have the ability to convert a variable-length input into a fixed-length output, and vice-versa for the decoder. It is known that recurrent networks, such as LSTM [13] have the ability to compute a function of variable-length inputs [14], and that LSTM has also been used successfully in sequence-to-sequence tasks [12]. Therefore in this paper LSTM is used as the encoder and decoder. LSTM is a type of recurrent neural network which was specifically designed to handle the vanishing and exploding gradients problems. Let $h_t^{(k)}$ denote the hidden states of the k^{th} layer of a multi-layer LSTM, and the input x_t can be denoted by $h_t^{(0)}$, in which the input vector is d-dimensional while the hidden states are p-dimensional. We also denote $c_t^{(k)}$ as the *cell state*, which is an additional p-dimensional hidden vector. The cell state can be considered as a type of long-term memory. The update matrix denoted by $W^{(k)}$ with size $4p \times 2p$ is used in the update process given the inputs $[h_t^{(k-1)}, h_{t-1}^{(k)}]$. There are four intermediate, p-dimensional vector variables used in the update process, i (input), f (forget), o (output), and c, which are all $4p$-dimensional vectors. The equations for the update process are as follows (where *sigmoid* and *tanh* denotes the sigmoid and tanh function respectively, and $\odot$ denotes the element-wise product operator):

$$\begin{bmatrix} i \\ f \\ o \\ c \end{bmatrix} = \begin{pmatrix} sigmoid \\ sigmoid \\ sigmoid \\ tanh \end{pmatrix} W^{(k)} \cdot \begin{bmatrix} h_t^{(k-1)} \\ h_{t-1}^{(k)} \end{bmatrix} \qquad (3)$$

$$c_t^{(k)} = f \odot c_{t-1}^{(k)} + i \odot c, h_t^{(k)} = o \odot tanh(c_t^{(k)})$$

The sigmoid and tanh functions are defined as follows:

$$sigmoid(x) = \frac{1}{1+e^{-x}}, tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (4)$$

After the cold start procedure is completed, the surrogate model G will be used instead of training the DNN candidate architectures. However, in order to ensure that the surrogate model retains its accuracy, it is necessary for it to be updated

periodically. In this paper, the surrogate model is updated periodically after every N_{update} generations. More specifically, let Gen be the current generation ($N_{coldstart} < Gen \leq N_{gen}$). If $(Gen - N_{coldstart})\%N_{update} \neq 0$ then the surrogate model is used to predict the accuracy of each DNN architecture x^{var} in the current generation, otherwise each DNN in the current generation x^{var} will be trained and evaluated on the training set to find its performance metric y. The encoder is used to find the fixed-length representation $x^{fixed} = Enc(x^{var})$ and $\{x^{fixed}, y\}$ will be added to $\mathbf{D}_{surr}$. The surrogate model will be updated at the end of the generation using Equation 2.

Our proposed method is described in Algorithm 1. The algorithm receives as inputs the training set $\mathbf{D}$, validation set $\mathbf{V}$, number of candidates N_{cand}, number of generations N_{gen}, number of cold start generations $N_{coldstart}$, the surrogate model G and the number of update generations for the surrogate model N_{update}. In lines 1-2, the cold start and the surrogate data is initialized as the empty set, and the candidates are initialized randomly. Lines 3-8 denotes the cold start procedure, in which each DNN candidate during the first $N_{coldstart}$ generations are trained on $\mathbf{D}$ and its performance is evaluated on $\mathbf{V}$, and the DNN encoding along with the performance is added to $\mathbf{D}_{coldstart}$. Afterwards, in line 9, the encoder-decoder is trained on $\mathbf{D}_{coldstart}$, then in line 10, the encoder is used to convert the variable-length encodings in $\mathbf{D}_{coldstart}$ into a fixed-length vector, which are then added to $\mathbf{D}_{surr}$. In line 11, the surrogate model G is trained on the surrogate data $\mathbf{D}_{surr}$. In lines 12-25, the algorithm continues from generation $N_{coldstart} + 1$ to the last generation. After each N_{update} generations, the DNN candidates are trained normally and then the surrogate model is updated using the new results (lines 19-22). Otherwise, for each DNN candidate, the encoder is first used to convert its encoding into a fixed-length representation, which is then used by the surrogate model to predict its performance without training (lines 14-17). Finally, in line 26, the optimal DNN architecture found during the evolutionary process is returned.

Algorithm 1 Training process

Input: Training set $\mathbf{D}$, validation set $\mathbf{V}$, number of candidates N_{cand}, number of generations N_{gen}, number of cold start generations $N_{coldstart}$, the surrogate model G, the number of update generations for the surrogate model N_{update}

Output: The best DNN architecture found during the evolutionary process.

1: $\mathbf{D}_{coldstart} \leftarrow \emptyset, \mathbf{D}_{surr} \leftarrow \emptyset$
2: Initialize the candidates $\{x_n^{var}\}_{n=1}^{N_{cand}}$
3: **for** $Gen \leftarrow 1$ to $N_{coldstart}$ **do**
4: **for** $n \leftarrow 1$ to N_{cand} **do**
5: Train the DNN represented by x_n^{var} on $\mathbf{D}$ and evaluate its performance y_n on $\mathbf{V}$, and add $\{x_n^{var}, y_n\}$ to $\mathbf{D}_{coldstart}$
6: **end for**
7: Perform evolutionary operators on $\{x_n^{var}\}_{n=1}^{N_{cand}}$ to generate the new population
8: **end for**
9: Train the encoder-decoder on $\mathbf{D}_{coldstart}$ using Equation 1.
10: Create $\mathbf{D}_{surr}$ by using the encoder Enc to convert each variable-length encoding $x^{var} \in \mathbf{D}_{coldstart}$ into a fixed-length representation, i.e. $x^{fixed} = Enc(x^{var})$
11: Train the surrogate model G on $\mathbf{D}_{surr}$
12: **for** $Gen \leftarrow N_{coldstart} + 1$ to N_{gen} **do**
13: **if** $(Gen - N_{coldstart})\%N_{update} \neq 0$ **then**
14: **for** $n \leftarrow 1$ to N_{cand} **do**
15: $x_n^{fixed} = Enc(x_n^{var})$
16: $y_n = G(x_n^{fixed})$
17: **end for**
18: **else**
19: **for** $n \leftarrow 1$ to N_{cand} **do**
20: Train the DNN represented by x_n^{var} on $\mathbf{D}$ and evaluate its performance y_n on $\mathbf{V}$
21: **end for**
22: Update the surrogate model G
23: **end if**
24: Perform evolutionary operators on $\{x_n^{var}\}_{n=1}^{N_{cand}}$ to generate the new population
25: **end for**
26: **return** The best DNN architecture found during the evolutionary process.

IV. Experimental Studies

A. Experimental Settings

We applied our proposed method to a well-known evolutionary DL method called EvoCNN[1] [7] which uses an EA to evolve CNNs. We experiment on several datasets: FashionMNIST [8], MNIST, MNIST with Rotated Digits (MRD) [32] like in experiments of EvoCNN [7]. The FashionMNIST and MNIST datasets have 50,000 training images and 10,000 test images, while the MNIST with Rotated Digits dataset has only 12,000 training images and 50,000 test images, and is more challenging for machine learning algorithms than the original MNIST dataset. The number of generations N_{gen} and number of candidates N_{cand} were set to 100, the number of cold start generations $N_{coldstart}$ and the number of update generations N_{update} were set to 5. A LSTM layer was used for the encoder and decoder, with the fixed-length vector having a size of 100. The training of the encoder-decoder was done for 300 epochs using the Adam algorithm. We compared the results of the original EvoCNN method and our proposed variable-length encoding based surrogate model based on two performance metrics namely accuracy and F1 score. We also compare the optimisation time between the two cases. Two surrogate models used in our experiments were Random Forest and XgBoost. These models were chosen because they are well-known surrogate models in the literature and they generally provide good results. The hyperparameters for the EvoCNN algorithm were set in the same way as [7].

[1]https://github.com/yn-sun/evocnn

B. Results and Discussions

In this section, we compared the results between the original method (no surrogate model was used), and when two surrogate models were used. Figure 2 shows the results of the original EvoCNN method, and when Random Forest and XgBoost were used as the surrogate model on the FashionMNIST dataset. It can be seen that the performances of the original

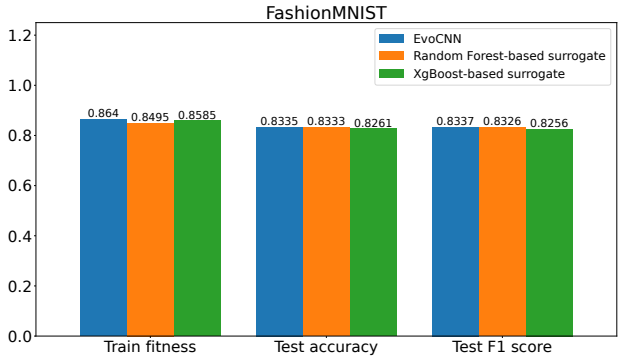

Fig. 2. The results on the FashionMNIST dataset

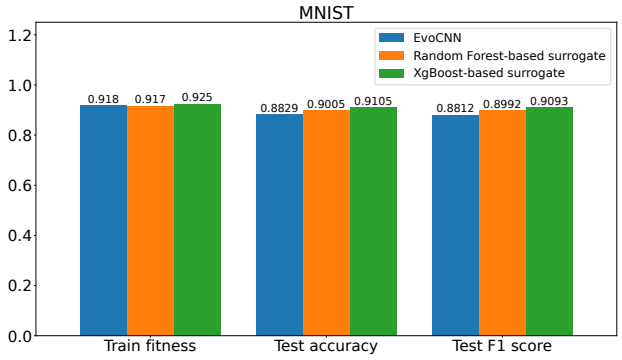

Fig. 3. The results on the MNIST dataset

method and the two surrogate models are roughly similar. The train fitness of the original method is 0.864, followed by XgBoost at 0.8585 and Random Forest at 0.8495. However, for the test accuracy and F1-score, both the original method and Random Forest scores at around 0.83, while XgBoost only obtains a score of around 0.82.

Figure 3 shows the experimental results on MNIST dataset. For this dataset, XgBoost achieved the best score for both the train fitness and the test scores. The best train fitness score was obtained by XgBoost at 0.925, while the scores for both EvoCNN and Random Forest were around 0.917. For the test accuracy, the best score was achieved by XgBoost at 0.9105, which is higher than Random Forest and the original method by 1% and 2.76% respectively. Similarly, XgBoost also obtained the best F1-score on the test set at 0.9093, followed by Random Forest at 0.8992 while the original method only scores 0.8812.

Figure 4 shows the results for the MNIST Rotated Digits dataset. It can be seen that the results are not as good as the previous datasets, due to the fact that this dataset only contains 12,000 training images compared to 50,000 training images for the other two datasets, and that the images in this dataset have been chosen to increase the difficulty for classification algorithms. XgBoost achieved the highest train fitness at 0.6835, while the original method and Random Forest only obtains around 0.58-0.59. On this dataset, EvoCNN

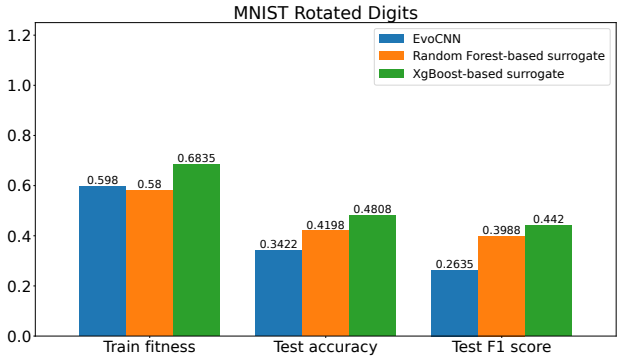

Fig. 4. The results on the MNIST Rotated Digits dataset

performed poorly, achieving the accuracy at 0.3422. Two surrogate models meanwhile made the results better, obtaining 0.4198 and 0.4808 of Random Forest and XgBoost respectively. A similar situation can be seen for the test F1-score, in which XgBoost obtained the highest F1-score at 0.442, followed by Random Forest at 0.3988, while the F1-score of the original method was only around 0.26. The experimental results indicated that using the proposed surrogate model can achieve competitive performance while significantly reducing the time of optimisation process compared to using original method.

Figure 5 shows the confusion matrices of EvoCNN and our proposed method on the FashionMNIST, MNIST and MNIST Rotated Digits datasets (from left to right, top to bottom). Each row denotes the results on each dataset (FashionMNIST, MNIST and MNIST Rotated Digits) and for each row, each column denotes the results of the original method, Random Forest and XgBoost respectively. For the 1^{st} row (FashionMNIST), it can be seen that the main misclassification types are the same for all three cases (original method, Random Forest and XgBoost). The original method mostly misclassified class 6 as class 0 (187 cases), class 2 (97 cases), class 4 (84 cases), and misclassified class 0 as class 6 (84 cases), class 2 as class 6 (133 cases), class 4 as class 6 (120 cases), class 4 as class 2 (100 cases) and class 2 as class 4 (142 cases). Both Random Forest and XgBoost have less images of class 2 and 4 being misclassified as class 6 compared to the original method (around 80-90 for both compared to more than 120 for the original method). On the other hand, XgBoost and Random Forest misclassified more images of class 6 as class 2 and class 4 compared to the original method, which is from 80 to around 100. XgBoost also wrongly classified 40 images of class 9 as class 5, as opposed to just 6 and 19 images by the original method and Random Forest.

With respect to the 2^{nd} row (MNIST), the original method misclassified 119 images of character 4 as character 9, which is almost twice as those of Random Forest (51) and XgBoost (67). The highest misclassification cases for each category of both Random Forest and XgBoost are much smaller than 100, as opposed to the original method (119). Both Random

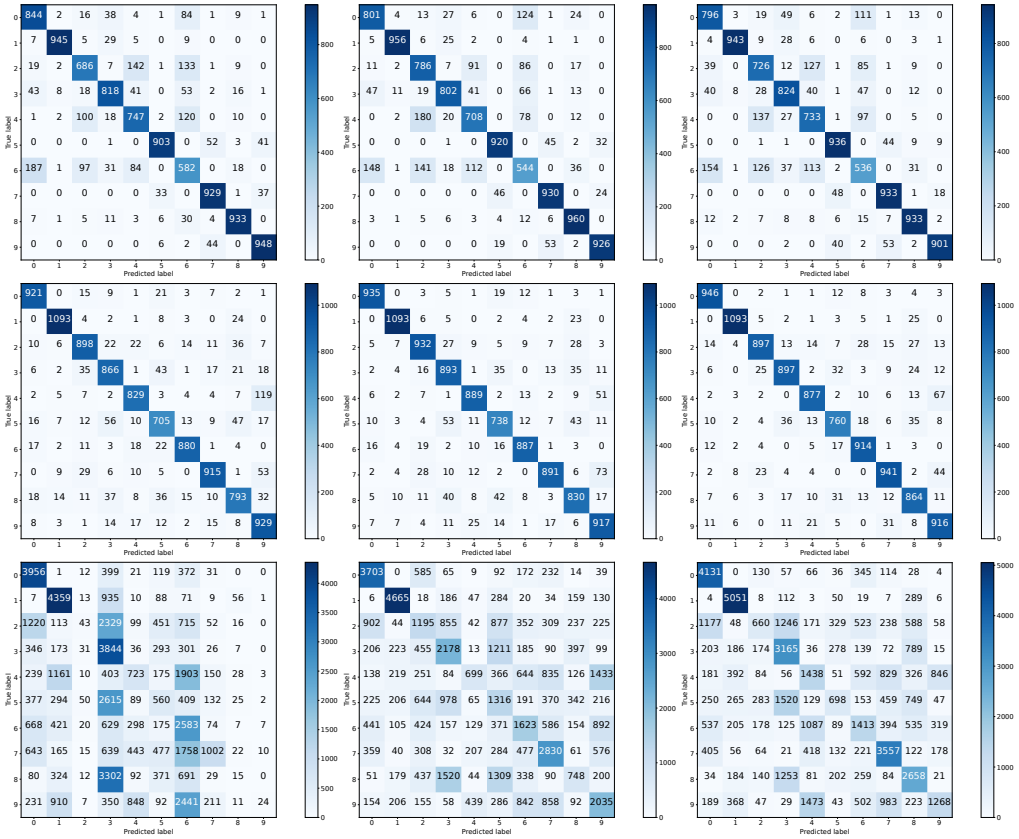

Fig. 5. Confusion matrices of EvoCNN (first column), and our proposed method when Random Forest (second column) and XgBoost (third column) are used on the FashionMNIST, MNIST and MNIST Rotated Digits datasets (top to bottom rows respectively)

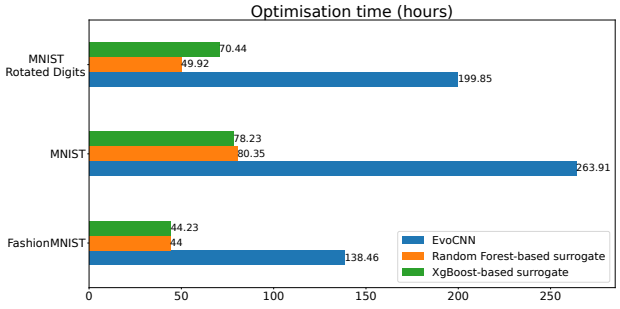

Fig. 6. Optimisation time of EvoCNN and the surrogate-based methods using Random Forest and XgBoost.

Forest and XgBoost had 55 misclassification entries which are smaller than 10, compared to only 49 entries for the original method. The average misclassification number for each entry by the original method was 13.01, which is higher than those of Random Forest and XgBoost by 1.95 and 3.07 respectively. The number of correct classification cases for each class by Random Forest and XgBoost were usually higher than that of the original method by 30, in which the highest result are between XgBoost and the original method for character 8 at 71.

For the 3^{rd} row (MNIST Rotated Digits), it can be seen that there are many more misclassification cases compared to the other datasets. The original method performed poorly on this dataset, with very low correct classification for character 2 (43), character 8 (15) and character 9 (24). The highlight misclassification types are mistaking character 8 as character 3 (3302 instances), character 9 as character 6 (2441 instances) and character 2 as character 3 (2329 instances). On the other hand, there are only few instances being misclassified as character 7-9. In contrast, the highest misclassification numbers for each category by both Random Forest and XgBoost were 1520, which is around half of that of the original method. The original method made on average 365.46 mistakes for each category, while Random Forest and XgBoost only makes 322.31 and 288.46 misclassifications for each category on average. For Random Forest, the highest misclassification numbers were 1520 (character 8 misclassifed as character 3), 1433 (character 4 misclassified as character 9) and 1309 (character 8 misclassified as character 5). Therefore, it can be seen that for all datasets, our proposed variable-length based surrogate model has less misclassified instances compared to the original method.

Figure 6 shows the optimisation time of original method (EvoCNN) and our proposed surrogate-based models using Random Forest and XgBoost. It can be seen that the surro-

gate models reduce the computational time by several orders of magnitude compared to EvoCNN. For the FashionNIST dataset, while EvoCNN took 138.46 hours, Random Forest-based and XgBoost-based surrogate model took only 44 hours and 44.23 hours, respectively, which is 3-time faster than EvoCNN. Similarly, for the MNIST dataset, EvoCNN took 263.91 hours, compared to just around 80.35 and 78.23 hours by Random Forest and XgBoost-based surrogate model respectively. For the MNIST Rotated Digits dataset, EvoCNN required 199.85 hours for training, while Random Forest and XgBoost-based surrogate model only required 49.92 and 70.44 hours respectively. These results indicate that our proposed surrogate model significantly reduces computational time while obtaining similar performance compared to EvoCNN.

V. Conclusion

In this paper, we introduced a novel surrogate model for variable-length encoding to optimise deep learning architecture. An encoder-decoder model was used to convert the variable-length encoding of the DNN into a fixed-length representation, which allows the surrogate to predict the DNN performance without training. During the initial cold-start stage, its weights are trained by matching the output as closely to the input as possible. Afterwards, the decoder is discarded and given a variable-length DNN encoding, the encoder transforms this encoding into a fixed-length vector, which will then be used by a surrogate model to predict the DNN performance without training. A LSTM was used as the encoder and decoder, and experiments were conducted based on a popular evolutionary DL method. We compared the results using two types of surrogate models namely Random Forest and XgBoost. The results indicated that our proposed method maintain competitive accuracy while reducing computational time significantly.

VI. Acknowledgement

This work was supported by the Scottish Government through The RSE Scotland Asia Partnerships Higher Education Research (SAPHIRE) Fund [grant number 2970].

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[2] N. Li, L. Ma, G. Yu *et al.*, "Survey on evolutionary deep learning: Principles, algorithms, applications, and open issues," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–34, 2023.

[3] B. Zoph, V. Vasudevan, J. Shlens *et al.*, "Learning transferable architectures for scalable image recognition," in *Proceedings of CVPR*, 2018, pp. 8697–8710.

[4] A. Zela, A. Klein, S. Falkner *et al.*, "Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search," *ICML AutoML Workshop*, 2018.

[5] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," *Proceedings of ICLR*, Apr. 2019.

[6] M. Baldeon Calisto and S. K. Lai-Yuen, "AdaEn-Net: An ensemble of adaptive 2D-3D Fully Convolutional Networks for medical image segmentation," *Neural Networks*, vol. 126, pp. 76–94, Jun. 2020.

[7] Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. PP, 10 2017.

[8] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[9] C. He, Y. Zhang, D. Gong *et al.*, "A review of surrogate-assisted evolutionary algorithms for expensive optimization problems," *Expert Systems with Applications*, vol. 217, p. 119495, 2023.

[10] Q. Gu, Q. Wang, X. Li *et al.*, "A surrogate-assisted multi-objective particle swarm optimization of expensive constrained combinatorial optimization problems," *Knowl.-Based Syst.*, vol. 223, p. 107049, 2021.

[11] Z. Hong, M. Tao, L. Liu *et al.*, "An intelligent approach for predicting overbreak in underground blasting operation based on an optimized xgboost model," *Eng. Appl. Artif. Intell.*, vol. 126, p. 107097, 2023.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of NIPS*, 2014, p. 3104–3112.

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

[14] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Springer, 2018.

[15] D. Ezzat, A. E. Hassanien, and H. A. Ella, "An optimized deep learning architecture for the diagnosis of covid-19 disease based on gravitational search optimization," *Applied Soft Computing*, vol. 98, p. 106742, 2021.

[16] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "GSA: a gravitational search algorithm," *Inf. Sci.*, vol. 179, no. 13, pp. 2232–2248, 2009.

[17] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[18] R. Miikkulainen, J. Z. Liang, E. Meyerson *et al.*, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017.

[19] E. Real, A. Aggarwal, Y. Huang *et al.*, "Regularized evolution for image classifier architecture search," in *Proceedings of AAAI*, 2019.

[20] X. Wang, Y. Jin, S. Schmitt *et al.*, "An adaptive bayesian approach to surrogate-assisted evolutionary multi-objective optimization," *Inf. Sci.*, vol. 519, pp. 317–331, 2020.

[21] C. Chen, X. Wang, H. Dong *et al.*, "Surrogate-assisted hierarchical learning water cycle algorithm for high-dimensional expensive optimization," *Swarm and Evolutionary Computation*, vol. 75, p. 101169, 2022.

[22] R. G. Regis, "Evolutionary programming for high-dimensional constrained expensive black-box optimization using radial basis functions," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 326–347, 2013.

[23] R. Luo, F. Tian, T. Qin *et al.*, "Neural architecture optimization," in *Proceedings of NIPS*, 2018, p. 7827–7838.

[24] D. Lim, Y. Jin, Y.-S. Ong *et al.*, "Generalizing surrogate-assisted evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 329–355, 2009.

[25] H. Yu, Y. Tan, C. Sun *et al.*, "A generation-based optimal restart strategy for surrogate-assisted social learning particle swarm optimization," *Knowl.-Based Syst.*, vol. 163, pp. 14–25, 2019.

[26] P. Liao, C. Sun, G. Zhang *et al.*, "Multi-surrogate multi-tasking optimization of expensive problems," *Knowl.-Based Syst.*, vol. 205, p. 106262, 2020.

[27] S. Chu, Z. Yang, M. Xiao *et al.*, "Explicit topology optimization of novel polyline-based core sandwich structures using surrogate-assisted evolutionary algorithm," *Comput. Methods Appl. Mech. Eng.*, vol. 369, p. 113215, 2020.

[28] T. Zhang, C. Lei, Z. Zhang *et al.*, "AS-NAS: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning," *IEEE Trans. Evol. Comput.*, vol. 25, no. 5, pp. 830–841, 2021.

[29] Z. Lu, S. Liang, Q. Yang *et al.*, "Evolving block-based convolutional neural network for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–21, 2022.

[30] K. Han, T. Pham, T. H. Vu *et al.*, "VEGAS: a variable length-based genetic algorithm for ensemble selection in deep ensemble learning," in *Proceedings of ACIIDS*, 2021, pp. 168–180.

[31] Y. Gong, Y. Sun, D. Peng *et al.*, "Bridge the gap between fixed-length and variable-length evolutionary neural architecture search algorithms," *Electronic Research Archive*, vol. 32, no. 1, pp. 263–292, 2024.

[32] H. Larochelle, D. Erhan, A. Courville *et al.*, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of ICML*, 2007, p. 473–480.