Evolving Neural Networks for Text Classification using Genetic Algorithm-based Approaches

Hayden Andersen
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
andershayd@ecs.vuw.ac.nz

Sean Stevenson
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
sean.stevenson93598@gmail.com

Tuan Ha
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
tobias.tha@outlook.com

Xiaoying Gao
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
xiaoying.gao@ecs.vuw.ac.nz

Bing Xue
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
bing.xue@ecs.vuw.ac.nz

2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9504920

***Abstract*—Convolutional Neural Networks (CNNs) have been well-known for their promising performance in text classification and sentiment analysis because they can preserve the 1D spatial orientation of a document, where the sequence of words is essential. However, designing the network architecture of CNNs is by no means an easy task, since it requires domain knowledge from both the deep CNN and text classification areas, which are often not available and can increase operating costs for anyone wishing to implement this method. Furthermore, such domain knowledge is often different in different text classification problems. To resolve these issues, this paper proposes the use of Genetic Algorithm to automatically search for the optimal network architecture without requiring any intervention from experts. The proposed approach is applied on the IMDB dataset, and the experimental results show that it achieves competitive performance with the current state-of-the-art and manually-designed approaches in terms of accuracy, and also it requires only a few hours of training time.**

***Index Terms*—Genetic Algorithm, Convolutional Neural Network, Text Classification**

I. Introduction

Text classification is to automatically categorise text documents into semantically related groups [1]. Effective text classification is important given that the number of digital documents is growing exponentially in the real world. Current state-of-art techniques for text classification are deep learning models such as *CNNs (Convolutional Neural Networks)* [2], [3]. The performance of CNNs depends heavily on their architectures and hyperparameter settings, which are mainly designed in a manual way in almost state-of-the-art CNNs. Not only does this manual process require domain knowledge, but it is also time-consuming and error-prone. Therefore, neural architecture search has emerged to automatically find the optimal architecture of CNN on a given task.

Evolutionary Computation (EC) is a population-based search technology that has shown promise in neural architecture search for image classification [4], [5]. However, there has not been much work on using EC on neural architecture search for text classification. Therefore, this paper aims to use EC techniques to automatically search for the network architecture along with optimising the hyperparameters for the text classification task. We expect this approach to be able to (i) find optimal network architectures and hyperparameters and (ii) improve the text classification performance. One typical EC method is *Genetic Algorithm (GA)*, which emulates the process of natural selection to evolve a population of candidate solutions. Better solutions have a higher chance of being chosen as parents to produce offspring to form the population for the next generation, using mainly the crossover and mutation operations; this results in populations with better results (fitness values) overall. There has been research on using GAs for evolving CNN architectures [6]–[8], but almost all of them are applied on image classification. There is very limited research on evolving CNNs for text classification.

This paper aims to investigate the performance of GAs on evolving CNNs for text classification. To achieve the goal, *two* different representation methods are proposed. The first encodes a more traditionally-designed CNN, mainly evolving hyperparameters within the network to fine-tune a network for each specific problem. The second method evolves the network structure itself, allowing for networks that are designed very differently than what one would create by hand. The second method is partially inspired by *EXACT* [9], a method to evolve networks for image classification.

The rest of this paper is organised as follows. Section II outlines the background and summarises the related work. Section III and IV describe our two proposed methods. Section V details the experimental setup and the results. The conclusion and future work are given in Section VI.

II. Background

This section starts with a brief introduction on related research areas including *word embeddings* in text classification,

CNNs, GAs and then summarises recent work on evolving neural networks.

A. Word Embeddings

While many different methods have been used for text representation in text classification, word embeddings have shown extremely competitive results in the last decade, while requiring relatively little processing power once trained [10]. In word embeddings, each document is represented as an $m \times n$ matrix $\boldsymbol{D}$, where m is the maximal number of words in a sequence, and n is the word embedding dimensions. Each row of this matrix is a dense vector $\boldsymbol{w_i} \in \mathbb{R}^n$ which indicates the position of the word within the vector space, called word embeddings. Word embeddings can be either trained using the input corpus or generated using pre-trained word embeddings such as *GloVe* [11] and *Word2Vec* [10]. Mathematically, by using word embeddings, the vector differences between $man - woman$ and $king - queen$ are roughly equal. In this paper, we use the GloVe [11] word embeddings which are pre-trained based on the idea that ratios of word to word co-occurrence are more likely to encode some sort of meaning. The particular GloVe representation used in this paper is trained on Wikipedia and Gigaword [1] with 50 dimensions.

B. Convolutional Neural Networks

CNNs are a class of *Neural Networks* that achieve data transformation on the input data (in this case a text corpus) via a number of layers/operations, produces an output that can be used to classify the data. In this paper, the operations that are considered are *convolution*, *pooling*, and *dense connections*.

In *Convolution* layers, one or more filters slide over the input data, summing the product of each value in the input and the value in its corresponding position in the filter. This value then gets placed as part of the output data. After that, the filter slides across an amount based on its stride value (in this case, the number of rows shifted), and repeats until the end of the input data is reached. In this work, all stride values are set to 1. An example of using CNN and word embeddings is given in Fig. 1.

Pooling operations are very similar to convolution operations; however, their purpose is to simply reduce the size (and therefore the complexity) of the input data. The method of pooling used in the paper is max pooling, which takes the highest value in the filter at each point and sets the output to that value. Max pooling has been proven to be the best pooling operator when it comes to text classification [12].

Dense layers simply connect each value in the input data to each value in the output data, multiplied by the weight of each connection.

Both the convolution and dense layers apply an activation function to the output before becoming the input of the next layer. In this paper, *ReLU* [13] activation is used. This is an activation function that simply sets all negative values to *zero* while retaining positive values linear to the value before activation.

[1]https://catalog.ldc.upenn.edu/LDC2011T07

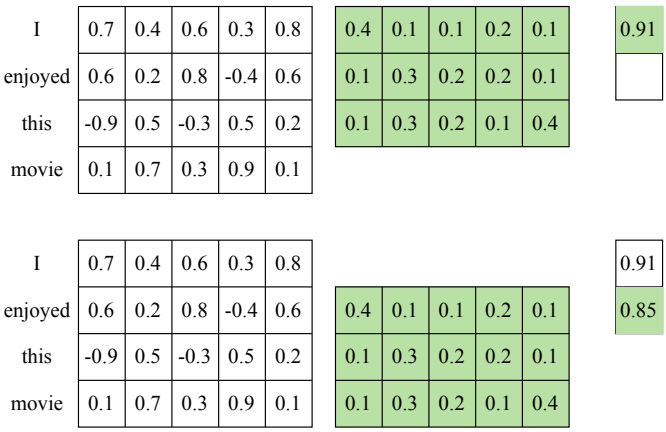

Fig. 1. Example of using CNN and word embeddings. In this example, $m = 4$, $n = 5$, $stride = 1$, and $size_{filter} = 3$.

Additionally, CNNs for text classification have a few additions. Before the input is passed into the full network, it is put into an *embedding layer* that applies the text embedding to the input in order to create the encoded vectors. As stated above, this paper uses the pre-trained GloVe embeddings [11]. At the end of the network, the output is flattened down to a single vector and passed through a final dense layer whose output size is equal to the number of classes. This is then passed through a *softmax* [14] activation function to give probabilities of the input belonging to each class.

C. Related Work

The majority of research in the field of using EC to automatically evolve neural networks is for image classification [15]–[17], and there is a lack of research on using such techniques for text classification.

Knippenberg et al. [15] proposed a neuro evolution-based approach to evolve CNNs. The initial conception of their idea was sourced from Large-scale evolution of image classifiers, which acted as a baseline of using genetic programming to evolve a CNN. They identified issues with how the selection process within the evolutionary process was conducted, which incurred a large computational cost. The method of Knippenberg et al. involved evolving a set of Convolutional *Autoencoders*, to find the best one to train a population of CNNs. The pairing of the most effective CNN and Autoencoder created the most optimal overall network. Autoencoders were used to reduce the input sample sizes. Their research showed the approach could reduce the training time by a factor of days.

Sun et al. [16] proposed a method using GA to evolve performant CNN architectures for image classification. The primary contributions were a new encoding scheme that extends the depths of CNNs by introducing skip connections as a layer that could be evolved, as well as running population evaluation across GPUs asynchronously. Firstly, the mutation operator as part of the CNN-GA strategy could add a skip or pooling layer, remove a layer or randomise the hyperparameter values of a block. Secondly, the crossover operator allowed

for variable-length individuals, meaning there was a higher ceiling on the potential evolutions per individual that could occur. As part of the algorithm, an environmental selection was employed, to filter the top individuals for the next generation. One benefit of the algorithm was that it required neither pre-processing nor post-processing. One aspect that was highlighted was the reduced probability of a pooling layer being evolved. Down-sampling affected the total information the system had to process, thus limiting the addition of this layer to the population might increase the performance. In contrast, there was a higher probability of mutating a skip layer, as this deepened the CNN to aid in the performance. Relative to state-of-the-art algorithms on selected datasets, their proposed algorithm outperformed almost all competitors.

Liang et al. [17] proposed LEAF, an Evolutionary AutoML Framework consisting of three main components: (1) the algorithm layer, (2) the system layer, and (3) the problem-domain layer. The first layer's purpose was to evolve the hyperparameters and architectures of the networks. The trained networks were then evaluated by the system layer, and their fitness values were received. For the problem-domain layer, it made use of both algorithm and system layers to solve the problems of hyperparameter optimisation, architecture search, and complexity minimization. LEAF was primarily applied to the image classification task, and it was also evaluated on the Wikipedia Detox dataset. However, to achieve state-of-the-art results, the running time of the method might take up to 2,000 hours.

Dufourq et al. [18] attempted to reduce the training time of neuro-evolutionary algorithms in their proposed method, *EDEN* (Evolutionary DEep Networks). The computational cost was reduced by decreasing the population size over generations and initialising with a small number of epochs for training; the number of epochs increased over generations. As a result, the method could remove networks with a poor performance quickly at the beginning of the algorithm, and then it could spend more effort into evaluating superior networks in later stages. This was also one of the first attempts to applying neuro-evolution for the sentiment analysis task. EDEN was applied on seven datasets (a combination of image and sentiment classification problems), and it achieved state-of-the-art results on three datasets.

III. Method One: GA1-CNN

This section introduces the main components of our first method using a GA-based approach to evolve the CNNs, named **GA1-CNN**.

A. Representation

Each individual is represented as a variable-sized vector. This vector can be any size $n \in \mathbb{N}^+$ that is a multiple of 3, encoding a network with $\frac{n}{3}$ layers. In this vector, each set of 3 indices together forms the representation of a single layer. The first value of the indices that falls within the predefined ranges determines the *layer type* out of convolution, pooling, dense, or deactivated (as shown in Table I). Deactivated layers will be ignored during the practice of constructing a neural network from the encoded vector.

TABLE I
Layer type ranges for architecture encoding of GA1-CNN method.

Layer	Lower bound	Upper bound
Convolution	$\geqslant 1$	$\leqslant 10$
Pooling	$\geqslant 11$	$\leqslant 20$
Dense	$\geqslant 21$	$\leqslant 30$
Deactivated	$\geqslant 31$	$\leqslant 40$

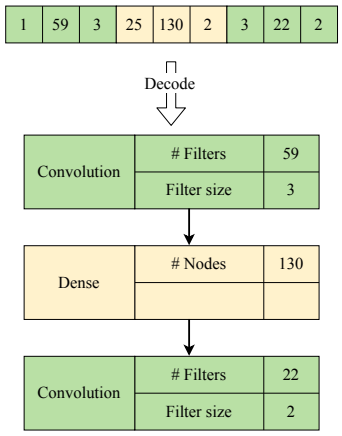

Fig. 2. An example of the representation of GA1-CNN and its decoded network architecture. This example has three layers: (1) a convolution layer, the number of filters is 59, the filter size is 3; (2) a dense layer with 130 nodes; (3) a convolution layer, the number of filters is 22, the filter size is 2.

The second value encodes the *hyperparameter* value of the layer. For convolution this is the *number of filters*, for pooling this is the *size of the pooling window*, for dense this is the *number of nodes*, and this value is unused for a deactivated layer.

The final value which encodes the *filter size* is only used for convolution layers. An example of this representation is illustrated in Fig. 2.

B. Initialisation

Each individual is initialised as a vector of size 9, representing a network with 3 layers. The first layer is always set to be a convolution layer. All parameters are set within the chosen ranges, with the layer type generated first and then the other two parameters of that layer generated based on the first value. This is repeated for each individual until the desired population size is reached.

C. Selection

We use *tournament selection* [19] as the selection operator to determines which individuals will be used for the mutation and crossover operators based on their fitnesses. In particular, a number of individuals are selected randomly according to the tournament size, and the mutation and crossover operators are then applied to the winners. In this work, the chosen tournament size is 3.

D. Mutation

A random number is generated between 0 and the number of layers currently in the individual. If this number is equal to the current number of layers then a new layer is added to the end of the individual, otherwise, the corresponding layer is replaced in the vector. A new layer is created by first generating a new layer representation value, then generating the two parameter values based on the layer type. After that, the three newly-created values are placed into the vector, or in the case of the final position appended to the vector in order to create a larger individual.

If the first layer is chosen, the layer type defaults to a convolution layer in order to prevent infeasible networks from being created.

E. Crossover

First, the network with a smaller number of layers $(depth)$ between the two individuals is found (the number of layers is calculated by dividing the size of the individual by 3). A random index is then generated in the range $[0, depth)$ and the two lists are swapped at that index, performing one-point crossover.

F. Evaluation

Evaluation is performed using the Python *Keras* [2] library, with a *Tensorflow* [3] back end. The model is built by first creating an embedding layer, using the pre-trained GloVe 50-dimensional vector embeddings. This layer is set to be trainable, in order to fine-tune the word embeddings for the target text corpus. Then, the layers encoded in the individual are added to the model. Finally, a flatten layer is added to condense the network down to a single vector, followed by a dense softmax layer to output the probabilities of each class. As an example, the full network created using the individual example (Fig. 2) is shown in Fig. 3.

Once the network has been created, it is trained on the *training* set, using the *Adam* [20] optimisation algorithm and calculating loss using *categorical cross-entropy*. The network is trained for a given number of epochs, with *early stopping* in place to prevent overfitting. A small subset of the training set is not used as training data and is used as an *validation* set to evaluate the performance of the trained network. If the loss on this data stays the same or gets worse for three epochs, the training of the network stops early.

For this method, we choose a fitness function that makes use of the *categorical cross-entropy loss* on the validation set. Once training is complete, the fitness of the individual is its best loss value on the validation set.

IV. Method Two: GA2-CNN

Our second method is also based on GA, but it uses a different representation which allows more flexible network architecture. The method is named **GA2-CNN**.

[2]https://keras.io/

[3]https://www.tensorflow.org/

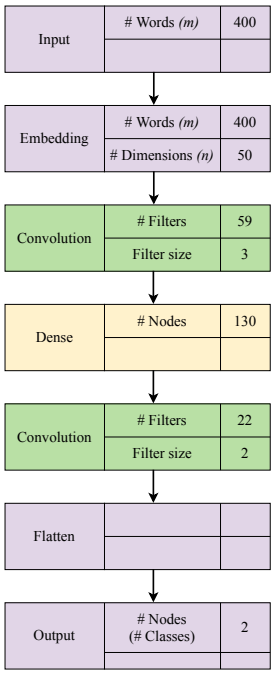

Fig. 3. A sample network evolved by the GA1-CNN method. The layers in grey background colour are always appended to the constructed neural networks.

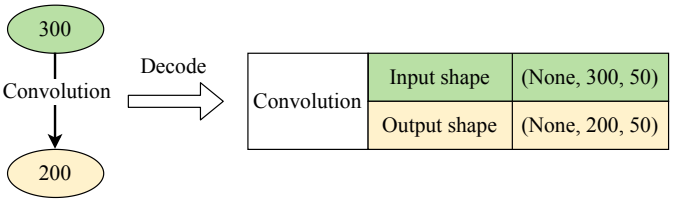

Fig. 4. A sample layer of the GA2-CNN method.

A. Representation

Each individual is represented in a tree-like graph, with *nodes* representing different size values, and *edges* representing the convolution or pooling layers. Nodes simply encode the size of input and output shapes of layers in the network, and edges encode the type of layers along with whether or not they are enabled in the network. For example, a convolution edge between two nodes of size 300 and 200 would encode the layer shown in Fig. 4. In the current implementation of the method, the number of filters of all convolution nodes is fixed to 50, which is also the number of dimensions of word embeddings.

B. Initialisation

All individuals share a base initialisation of an input node and an output node with an edge between them. In order to diversify the initial population, each individual then has a Split mutation applied to it (see Subsection IV-D).

C. Selection

Similar to GA1-CNN, we also use *tournament selection* (tournament size = 3) as the selection operator in this method to determines which individuals will be used for the mutation and crossover operators.

D. Mutation

We developed *seven* different mutations that can be applied to an individual as follows:

- *Split*: Edges can be split into two new edges, creating a new node between them and disabling the original edge. The newly created node has a random size between the two nodes it now connects.
- *Disable*: Edges can be disabled, preventing them from being counted for fitness evaluation of the individual.
- *Enable*: Disabled edges can be re-enabled in the individual.
- *Add*: New edges can be created between existing nodes, where an edge does not exist.
- *Change size*: The size of a node can be changed to a random value between the size of the nodes on either side, and in the case of the softmax node, any size between 1 and the smallest node above it.
- *Convert to pooling*: Any edges can be converted to a pooling edge, given there is enough difference in size between the nodes on either side. This mutation operator can also change the pooling size of an existing pooling edge. The pooling size of the edge is set to a random value $size_{pool} \in \mathbb{N}^+$ that satisfies Equation (1).

$$\frac{size_{in}}{size_{pool}} \geq size_{out} \tag{1}$$

- *Convert to convolution*: Any pooling edges can be converted to convolution edges.

E. Crossover

For crossover operation, GA2-CNN first checks which is the fitter individual between parents (see Subsection IV-F for more details about the fitness value of an individual). The method then adds all nodes in the parents to the child. The next step is to add the edges connecting nodes. For *enabled* edges, those which present in both parents are added first, and those which present in only one parent are added with a certain probability. This probability of the fitter parent is higher than that of the less fit parent, and hence, characteristics of good individuals have a higher chance to pass to the next generation. Finally, the remaining edges from each parent are added to the child as disabled edges. Fig. 5 demonstrates an example of the crossover operation of GA2-CNN method.

F. Evaluation

Evaluation is performed using the Python Keras library, with a Tensorflow backend. The model is built by first marking all nodes reachable through enabled edges from the input node, then building a CNN based on what marked nodes are reachable from the output node. The input node is converted to an embedding layer, and the output node is converted to a flatten layer, followed by a single dense softmax layer. All the edges in the network are converted to either convolution or pooling layers. Edges with *convolution* types are converted directly into convolution layers whose filter sizes are decided according to Equation (2).

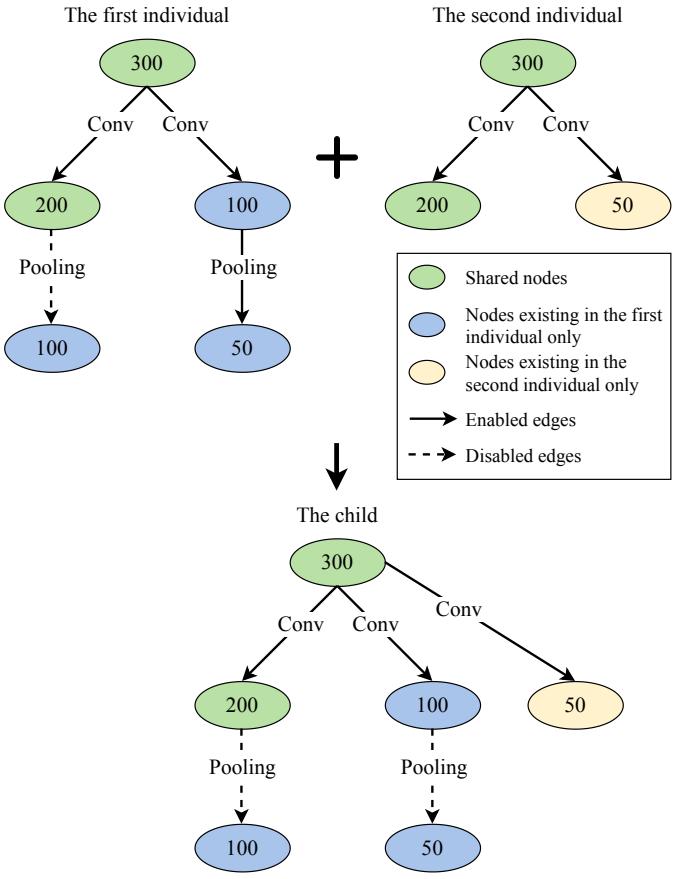

Fig. 5. An example of the crossover operation of GA2-CNN method. In this example, the first individual is the one with better fitness value.

For edges with *pooling* type, each of them are split into two layers: (i) a pooling layer with pooling size $size_{pool}$ and (ii) a convolution layer with filter sizes according to Equation (3).

$$size_{filter} = size_{in} - size_{out} + 1 \tag{2}$$

$$size_{filter} = floor(\frac{size_{in}}{size_{pool}}) - size_{out} + 1 \tag{3}$$

The purpose of adding an additional convolution layer in the edge with pooling type is to ensure that this part of the network can *always* be constructed, as long as this pooling edge satisfies Equation (1).

Any nodes in the individual with multiple inputs are turned into an *Add layer* [4] in order to merge the inputs; all nodes with only a single input simply continue to the next edge. *Add layer* is a layer that takes a list of tensors as input, all of

[4]https://keras.io/api/layers/merging_layers/add/

TABLE II
TRAINING HYPERPARAMETERS OF GA1-CNN.

Hyperparameter	Value
Crossover probability	0.3
Mutation probability	0.5
Population size	20
Generations	40
Evaluation epochs	10
Validation split	0.1

TABLE III
STRUCTURAL HYPERPARAMETER RANGES OF GA1-CNN.

Layer	Parameter Bounds	
Convolution	# Filters 1 to 128	Filter Size 1 to 5
Pooling	Pooling Size 1 to 5	- -
Dense	# Nodes 1 to 512	- -

the same shape, and returns a single tensor. Fig. 7 shows an example of a network containing an Add layer.

The created network is trained on the training set and is then evaluated on the validation set. For this method, we also choose the same fitness function that makes use of the *categorical cross-entropy loss* on the validation set as in GA1-CNN. Once the training is complete, the fitness of the individual is its best loss value on the validation set.

G. Cost Reduction

In order to reduce the computational cost, the network is only trained for a small number of epochs during each fitness evaluation, and we also use a small number of individuals. This reduces the amount of time needed to evaluate each individual, and the trend of training provides a good indicator of network rankings in comparison to each other.

V. EXPERIMENT SETUP AND RESULTS

GA1-CNN is run for 30 independent times, and the result is the average values of those runs. For each run, the number of generations is *40* and the number of individuals in the population is *20*; we use a small number of individuals to reduce the computational cost of the experiments. A record of the fittest individual is kept after each generation, saving the individual in case the algorithm is stopped partway through. At the end of the evolutionary process, the fittest individual is trained on the *training* set, using a subset of the training data for early stopping as was performed in evaluation. The final trained network is then evaluated on the *test* set, and the loss and accuracy of the network are recorded, along with how many epochs the network trained for before the early stopping occurred. Table II lists the training hyperparameters, whereas Table III shows the structural hyperparameter bounds used for the creation of the networks.

TABLE IV
TRAINING HYPERPARAMETERS OF GA2-CNN.

Hyperparameter	Value
Crossover probability	0.3
Mutation probability	0.5
Fitter parent inclusion	0.8
Less fit parent inclusion	0.2
Population size	20
Generations	30
Evaluation epochs	10
Validation split	0.1

TABLE V
MUTATION PROBABILITIES OF GA2-CNN.

Mutation	Probability
Disable edge	0.12
Enable edge	0.14
Split edge	0.17
Add edge	0.17
Change node size	0.12
Change edge to pooling	0.14
Change edge to convolution	0.14

GA2-CNN is run in a very similar fashion, except the algorithm is only run for *30* generations. The training hyperparameters used are shown in Table IV, and the probability of each mutation operation is shown in Table V.

Each method is run on a Cuda server, utilising an NVIDIA RTX 2070 to speed up the training process. Each method is run with the same seeds in order to give a fair evaluation between the two methods. Both methods are evaluated on the *IMDB* review polarity dataset [21]. This is a dataset of $25,000$ highly polar movie reviews for training and $25,000$ for testing. Of these reviews, half of them are *strongly positive* and half are *strongly negative*.

Our methods are compared with both manually-designed and automatically-designed approaches. The former includes *LSTMs* with implementation from [22] and *seq2-bown-CNN* [23], whereas the latter consists of *EDEN (Evolutionary DEep Networks)* [18], *T-NAML (Transfer Neural AutoML)* [24], and PSO-CNN (our variant implementation of GA1-CNN, but using PSO instead of GA).

A. Comparisons with peer competitors

According to the results in Table VI, the overall accuracy of the manually-designed approaches is higher than their automatically-designed counterparts. In particular, the former ranges from 0.865 to 0.923, whereas the latter from 0.774 to 0.881. For the compared automatic approaches, T-NAML still achieves state-of-the-art results; our proposed GA1-CNN method, whose accuracy is only lower than that of T-NAML by 0.011, ranks second. However, to reach that performance using one GPU, T-NAML method needs about 51 hours for training time [24], whereas our GA1-CNN method only needs 7 hours. For EDEN method, the execution time is 9 hours [18]. To summarise, one of our proposed methods achieves

TABLE VI
RESULTS OF THE COMPARED METHODS.

	Method	Accuracy
Manual Approaches	LSTM [22]	0.865
	LSTM initialized with word2vec [22]	0.900
	seq2-bown-CNN [23]	**0.923**
Automatic Approaches	EDEN [18]	0.858
	T-NAML [24]	**0.881**
	PSO-CNN	0.866
	GA1-CNN	0.870
	GA2-CNN	0.774

competitive performance with the current state-of-the-art and manually-designed approaches in terms of accuracy, and it also requires only a few hours of training time.

In addition, it is evident that the top results from automatically-designed approaches are still not able to outperform the manually-designed networks. However, it is worth noting that seq2-bown-CNN, the current state-of-the-art result on this dataset for manual methods, was introduced in 2015, and there is no significant improvement since then. On the other hand, T-NAML and our proposed methods are novel. To put it another way, the performance gap between manual and automatic approaches are shortening.

B. Comparisons between the two proposed methods

More detailed results of our proposed methods are shown in Table VII. From these results it can be observed that the accuracy of the simpler GA1-CNN is much better than that of GA2-CNN; however, the evolution time is about twice as long.

Examples of the networks created by GA1-CNN and GA2-CNN are given in Fig. 6 and Fig. 7 respectively. GA2-CNN creates many very similar networks in terms of size, while GA1-CNN has quite a lot of variety in the final network design. For GA1-CNN, some of the evolved networks are all convolution layers, some have multiple pooling layers, and some have multiple dense layers; all the evolved networks have a depth of around five.

Comparing the evolved network architectures, GA2-CNN tends to create networks with a small number of layers, although it has the potential to create much larger ones. When analysing the evolution logs, we find that GA2-CNN faces many difficulties to evolve into larger networks, due to larger ones beginning with larger loss values and therefore making them less likely to be chosen for passing on to the next generation. This could potentially be avoided by allowing multiple mutations per individual, but this would make it more difficult for the GA to narrow in on the best possible network. The more restrictive nature of GA1-CNN allows it to create a performant network more easily, since larger filter sizes in the case of GA2-CNN may reduce the fidelity in the information of the text document.

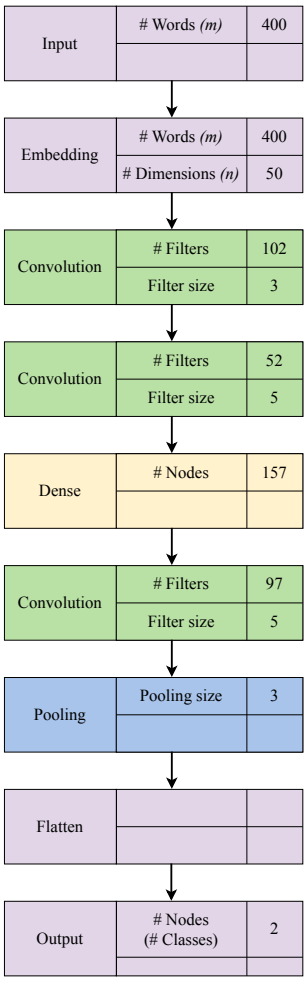

Fig. 6. Example result of GA1-CNN.

VI. CONCLUSION

This paper introduces two GA-based algorithms to automatically evolve CNN architectures for text classification. One of our methods can reach the performance of some of the manually-designed and fine-tuned systems. The main advantages of our systems are that they can evolve networks with flexible structures and that they can make use of different layer types. As a very early attempt of automatic methods for text classification, our research has shown promise in using EC to automatically evolve neural networks.

For future work, firstly, both methods can be run with more generations and with a larger population, in order to see if it converges to create more performant networks; GA2-CNN may be run with multiple mutations per individual in order to create larger networks. Secondly, the two proposed methods can be combined, using some of the concepts from both. If the evolution of GA2-CNN is constrained to only create new nodes that are slightly smaller than the previous ones, this would prevent the textual data from losing fidelity through each layer of the network. Last but not least, CNNs could also be integrated with other robust neural networks, such

TABLE VII
RESULTS OF THE PROPOSED METHODS.

Method	Accuracy	Std Dev	Evolution Time (hours)	Std Dev
GA1-CNN	0.870	0.008	7.192	2.049
GA2-CNN	0.774	0.009	3.778	0.209

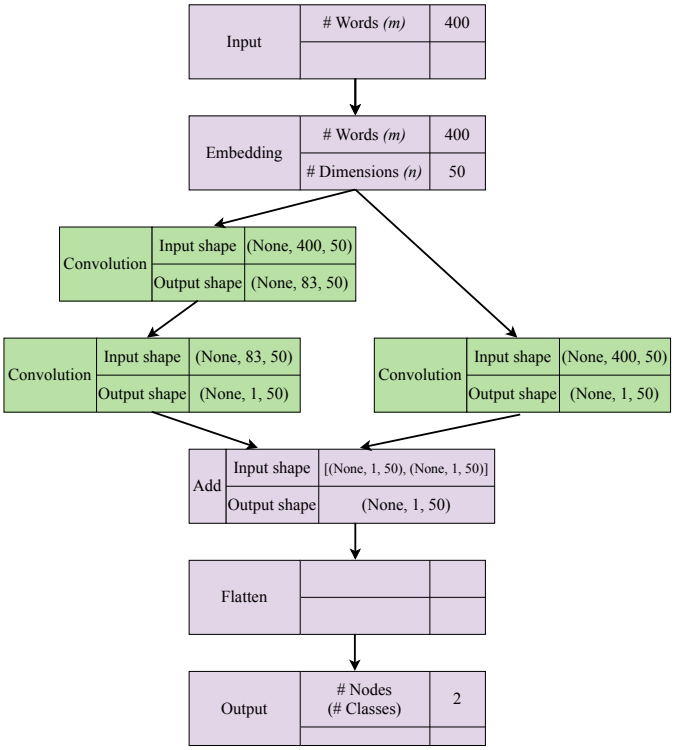

Fig. 7. Example result of GA2-CNN.

as *LSTM (Long Short-Term Memory)* which was shown to have excellent performance in dealing with text data. Evolving a CNN-LSTM structure in tandem, or just LSTM are both avenues that could be explored.

REFERENCES

[1] A. Hotho, A. Nürnberger, and G. Paaß, "A brief survey of text mining." in *Ldv Forum*, vol. 20, no. 1. Citeseer, 2005, pp. 19–62.

[2] T. Londt, X. Gao, and P. Andreae, "Evolving Character-level DenseNet architectures using genetic programming," *EvoApplications*, pp. 665–680, 2021.

[3] K. J. Madukwe, X. Gao, and B. Xue, "A GA-based approach to fine-tuning BERT for hate speech detection," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 2821–2828.

[4] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, "A survey on evolutionary machine learning," *Journal of the Royal Society of New Zealand*, vol. 49, no. 2, pp. 205–228, 2019.

[5] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2019.

[6] B. Ma, X. Li, Y. Xia, and Y. Zhang, "Autonomous deep learning: A genetic dcnn designer for image classification," *Neurocomputing*, vol. 379, pp. 152–161, 2020.

[7] Y. Sun, B. Xue, and M. Zhang, "Automatically evolving CNN architectures based on blocks," *CoRR*, vol. abs/1810.11875, 2018.

[8] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid ga-pso method for evolving architecture and short connections of deep convolutional neural networks," in *Pacific Rim International Conference on Artificial Intelligence*. Springer, 2019, pp. 650–663.

[9] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 127–128.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013*, 2013.

[11] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[12] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*, 2015.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[14] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[15] M. van Knippenberg, V. Menkovski, and S. Consoli, "Evolutionary construction of convolutional neural networks," in *International Conference on Machine Learning, Optimization, and Data Science*. Springer, 2018, pp. 293–304.

[16] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3840–3854, Sept. 2020, doi: 10.1109/TCYB.2020.2983860.

[17] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 401–409.

[18] E. Dufourq and B. A. Bassett, "Eden: Evolutionary deep networks for efficient machine learning," in *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*. IEEE, 2017, pp. 110–115.

[19] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex systems*, vol. 3, no. 5, pp. 493–530, 1989.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[22] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 3079–3087.

[23] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2015, pp. 103–112.

[24] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural automl," in *Advances in Neural Information Processing Systems*, 2018, pp. 8356–8365.