Automated Neural Network Construction with Similarity Sensitive Evolutionary Algorithms

Haiman Tian*, Shu-Ching Chen*, Mei-Ling Shyu† and Stuart H. Rubin‡
*School of Computing and Information Sciences
Florida International University, Miami, FL 33199, USA
Email: {htian005, chens}@cs.fiu.edu
†Department of Electrical and Computer Engineering
University of Miami, Coral Gabel, FL 33124, USA
Email: shyu@miami.edu
‡Space and Naval Warfare Systems Center Pacific
San Diego, CA 92152-5001, USA
Email: stuart.rubin@navy.mil

***Abstract*—Deep learning has been successfully applied to a wide variety of tasks. It generates reusable knowledge that allows transfer learning to significantly impact more scientific research areas. However, there is no automatic way to build a new model that guarantees an adequate performance. In this paper, we propose an automated neural network construction framework to overcome the limitations found in current approaches using transfer learning. Currently, researchers spend much time and effort to understand the characteristics of the data when designing a new network model. Therefore, the proposed method leverages the strength in evolutionary algorithms to automate the search and optimization process. Similarities between the individuals are also considered during the cycled evolutionary process to avoid sticking to a local optimal. Overall, the experimental results effectively reach optimal solutions proving that a time-consuming task could also be done by an automated process that exceeds the human ability to select the best hyperparameters.**

***Keywords*-deep learning; transfer learning; evolutionary algorithm; automated neural network construction; image classification**

I. Introduction

Classical Machine Learning (ML) techniques have achieved superior performance in many research domains for decades. Data modeling becomes challenging because uncertainties increase when applying ML to a broader area of study. Over the past several years, Deep Learning (DL) has overcome some of the limitations faced by classical ML for many research domains including visual data processing [1], speech recognition [2], and natural language processing [3]. For instance, when it comes to processing heterogenous data such as images and video, a lot of time is consumed by the feature engineering process to reduce the dimensionality of the data by identifying a cohesive subset of attributes that best represent the data. In essence, DL is a new technique that has proven to be appropriate in advancing the field of Artificial Intelligent (AI). It has considerably simplified the modeling process by incorporating both feature engineering and conceptual learning to directly process raw data then generate the final, conclusive results. Studies from different research fields have shown how deep learning eases the research work by requiring less task-specific manual process. Some notable frameworks that have leveraged deep learning into real-world applications include recommender systems, answer selection, and medical image analysis. Compared to the traditional independent feature engineering effort, deep learning models have better capability to generalize unseen combinations of features by embedding sparse inputs when solving large-scale regression and classification problems. Furthermore, traditional ML could not reach the same accuracy as the DL models in some cases. But developing and training a DL model from scratch is not always feasible for all researchers with limited access to computational facilities. Usually, training a robust deep neural network is a computationally expensive task that requires high-end Graphics Processing Units (GPUs) to perform the training process in a reasonable time. Moreover, recent work indicates that not all neurons are needed after the completion of the first iteration. This surprising result may lead to the constructive definition of the wiring pattern, which today is weighted through backprop and GAs. Fortunately, DL techniques are adaptable and transferable among different domains and applications. The rise in popularity of an optimization technique known as transfer learning [4], gave DL techniques the capability to influence more scientific research areas and solve their domain-specific problems. Practical usage of the features generated from well-designed pre-trained DL models has enhanced the performance of many applications. Those models are not only transferable to similar domains but also adaptable to different application fields. For example, the basic knowledge gained from a speech recognition task can now be easily applied to tasks in natural language processing [5].

Through the transfer learning process, traditional ML techniques can directly use the high-level semantic features learned from the DL models to perform many other domain-specific tasks while achieving higher performance than before. At the same time, constructing another comparatively simple deep neural network as the base network also appears to be a popular choice. When the new application domain is not very close to the source domain, deep features need to be transferred to better represent the targeted application. Still, even when adding just one layer to the pre-trained model, there is no guarantee that the new model will get promising results as expected in the source domain. Although we benefit in transfer learning, it still takes time to design a neural network after the feature extraction process – especially when the researcher has limited experience or knowledge of neural networks, the datasets, and the target tasks. To tackle this emerging issue, studies have recently focused on automating the network design process [6][7]. Two favorable directions to further explore are reinforcement learning [8] and Evolutionary Algorithms (EA) [9]. The latter has shown great promise in solving complex problems that the former has defined for several decades [10].

In this work, we aim to leverage the deep features from pre-trained Convolutional Neural Network (CNN) models in different applications without the need to spend most of the effort on examining the characteristics of each task. We propose a generalized framework to accommodate different datasets and problem domains. By integrating EA and other techniques to support the automated searching process, the hyperparameters of a new neural network, built for a specific task, are determined after the best individual is selected.

The remainder of this paper is organized as follows. In section II, the related work leverages evolutionary strategies to the DL field is provided. Section III explains the details of the proposed framework followed by a discussion of the experimental results in section IV. In section V, we offer conclusions.

II. Related Work

Many existing deep learning models have been successfully applied for different tasks. However, an automated approach to select the best model for each dataset and each domain is not available. To address this challenge, Long *et al.* [11] introduced Joint Adaptation Networks (JAN) that is based on a Joint Maximum Mean Discrepancy (JMMD) criterion to learn a transfer network by aligning multiple domain-specific layers (layer $fc7$ in AlexNet and layer $pool5$ in ResNet).

In [12], the authors proposed a Genetic Algorithm (GA) approach using transfer learning to enhance the performance of the CNN model in the image classification tasks. Deep features were generated from four pre-trained CNN models, which are ResNet50, Inception-v3, VGG16, and MobileNet. The experimental results showed that the proposed GA method can improve the performance of the baselines. However, while a straightforward genetic algorithm method can select the primary data representation model, it needs to be extended to enable deep neural network construction for specific tasks. Moreover, genetic algorithms will not scale here, or in natural evolution. What is needed are heuristic accelerators. Such heuristics can be learned and applied in a network configuration of neural networks. This provides coherency, a guiding necessary AI principle, and self-reference. The latter provides us with insight. Just as one of AIs failings led to the field of ML, so too does the failing of deep learning lead to the need for heuristics and heuristic acquisition. To fix the architecture of a hidden-layer neural network is to unnecessarily restrict that, which can and needs to be learned. Furthermore, it is argued that neural-based symbolic representations need to be enabled. It is well-known that modus ponens cannot be achieved without a symbolic representation. The creation of heuristics and their transfer-extension follows suit. The over-arching implication here is that today's deep learning architectures are not of sufficient Kolmogorov complexity to hold and learn to generalize strong knowledge. Both of these capabilities are inherent to not only real-world functionality, but commonsense reasoning as well. Commonsense reasoning has evaded capture by symbolic and neural AI alike. These are complex concepts; and, it will take some time to realize them in practice.

The authors in [13] discussed the advances in image classification with hyper-optimization. Computer clusters with large processing capacity GPUs allow trails and tests to be run. The researchers used hyper-optimization for training neural networks and deep belief networks, by optimizing hyperparameters with random searches and two greedy sequential methods. Sequential algorithms were applied to complex deep belief learning problems and improved results were obtained. The researchers validated the Gaussian Process Analysis (GPA) approach with a random sampling of the Boston housing data for a regression task. The dataset has 13 scaled input variables composed by 506 points to obtain a scalar regression output. An MLP network was trained with 10 hyperparameters. The hyperparameters included the hidden layer size, learning rate, iteration times, the Principal Component Analysis (PCA) preprocessing, and others. Sampling was used for the first 30 iterations, differentiated random samples were used for training, and the whole set up had 20 repetitions. Five GPUs were used; and, the test was run for 24 hours.

Recent research focuses on evolving the deep neural networks parameters or structures with GAs [14]. In [15], an improved genetic algorithm was proposed to tune the structure and parameters of a 3-layer Feed-Forward Network. Unlike deep neural networks that contain more complex structures, this network has a relatively simple structure which contains only one hidden layer. Therefore, there were few combinations of the available hyperparameters. So the best choice can be easily identified in advance.

In a recent work, Genetic CNNs [16] were proposed to learn the structure of deep neural networks automatically. The suggestion is to use GAs, since the network structures tend to rise exponentially with the number of layers. To serve this purpose, a new encoding scheme was suggested, which used a fixed-length binary sequence to indicate the network structure. Then, the accuracy on a reference set was used as the fitness function to determine the quality of each individual in a population. For each generation, the standard genetic operations were defined and these include the crossover and selection mutation needed to develop outstanding individuals while rejecting weaker ones. A standalone training method was used to identify the competitiveness. The genetic process was carried out on CIFAR10 with a small dataset to examine the capability to identify high quality structures. The output of the learned powerful structures was transferred to the ILSVRC2012 data that can be used for large visual recognition.

An alternative method of hyperparameter optimization for deep neural networks is presented in [17]. It compares the proposed approach, named Covariance Matrix Adaptation Evolution Strategy (CMS-ES), with the state-of-the-art Bayesian optimization algorithms for tuning hyperparameters of a CNN network. In their work, only two optimizers such as Adam and AdaDelt can be selected, which makes the expected performance more narrow.

III. Proposed Framework

An EA-based framework for automated neural network construction is illustrated in Figure 1. This framework aims to select the best network model for a specific task that uses transfer learning for image classification. Four major hyperparameters (i.e., number of neurons in one layer, number of fully connected layers in one model, the activation function, and the optimizer) are considered to formulate the network's gene. Specifically, a combination of those four hyperparameters composes a unique gene sequence that represents a network. The search process starts by randomly generating a group of networks as the initial population. Then, the initial networks evolve for several generations until they reach the end of the evolutionary limit. Several evolutionary strategies are used in this framework to improve the average performance in each population. Meanwhile, a Hamming distance matrix is calculated for every generation to evaluate the structural differences between each individual. Different genetic operations will be taken as reactions to the similarity evaluation, which ensures that the development of the new generation continues to cover a large searching space. The model that performs the best on the validation data is then identified at the end of the network's evolution. Next, a complete training process starts to build the final model for the targeting task.

A. Network Selection

Algorithm 1: Network Evolution

```
1  $RETAIN \leftarrow 0.4$, $SELECT \leftarrow 0.5$,
     $MUTATE \leftarrow 0.2$
2  for individual $i \in$ Population $p$ do
3      calculate FITNESS FUNCTION $f(i)$
4      $grade[i].score \leftarrow f(i)$
5  Sort $grade$ in descending order
6  for $u \in [0, grade.size - 1]$ do
7      $codes[u] \leftarrow$ENCODING $(grade[x].network)$
8      for $v \in [v + 1, grade.size - 1]$ do
9          $H_{uv} \leftarrow \sigma\ (codes[u], codes[v])$
10 $Significant = (\text{MAX}(H) - \text{MIN}(H))/2$
11 for $x \in [0, RETAIN * grade.size - 1]$ do
12     $parents.append(grade[x])$
13 # Random selection
14 for $x \in [RETAIN * grade.size, grade.size - 1]$ do
15     if $SELECT > random()$
16     AND $\forall H_{xs} > Significant$ WHERE
         $s \in parents$ then
17         $parents.append(grade[x])$
18 # Crossover
19 $size \leftarrow Population.size - parents.size$
20 while $children.size < size$ do
21     select $famale$ and $male$ randomly from $parents$
22     if $female \neq male$ then
23         $child = (male.partA + female.partB)$
24         if $MUTATE > random()$ then
25             MUTATE($child$)
26         $children.append(child)$
27     else
28         # Force Mutation
29         $child = male$
30         MUTATE($child$)
31         $children.append(child)$
32 $parents.append(children)$
33 return $parents$
```

Network selection starts with an initial population that is generated by random search. The process, as shown in Algorithm 1, takes all the networks (individuals) into the current generation to evolve. The network evolution incorporates all the genetic operations that might be triggered during the evolving process for every generation. The proposed evolutionary process enhances the operations in the traditional GA by controlling the similarities between the populations in subsequent generations. Specifically, it takes the strength of mutation operation in evolutionary programming to overcome the underlying weakness of crossover in the later generations. Force mutation and distance calculation ensure that the evolution process is capable of exceeding a local optimum in the searching space when the top networks

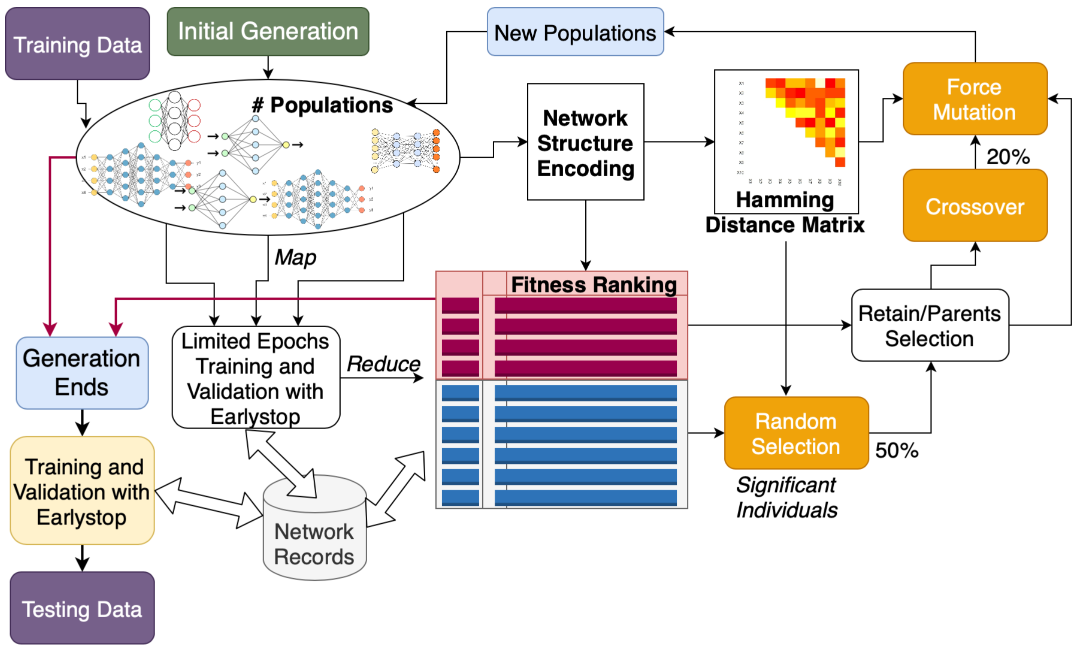

Figure 1. Proposed framework for automated neural network construction

in the same generation are very similar. In each generation, a portion of the top networks is selected as the parents to produce offspring that represent the new network structures. The selection is based on a specific retaining rate and a fitness function ranks the networks (lines 4 - 7). This fitness function is based on the formula of the averaged F1 score, which evaluates the performance of a specific network. Compared to using accuracy as the evaluation criterion, F1's score is more suitable for evaluating any dataset, whether balanced or imbalanced. The fitness function can be written as follows:

$$f(i) = (\sum_{c=1}^{C} \frac{2 * tP_c^i}{2 * tP_c^i + F_c^i})/C, \quad (1)$$

where C is the total number of classes in the targeting dataset, and i is the index of a unique network. tP is defined as the number of instances correctly identified as class c, and F is defined as the number of instances wrongly classified as c or others respectively for the network i.

Besides updating the ranking list of networks with their performance in each evolution, we use additional storage to record the information of the networks that have already appeared in previous generations. As we care only about the best performance that we have gotten for each unique gene sequence, only the highest fitness score for one combination of the hyperparameters will be stored for later references. Multiple networks that share the same gene, however, can appear in the ranking list, which represents the overall performance of the current population. The more times this structure is selected, the greater the chance that

Table I
THE AVAILABLE CHOICES FOR NETWORK HYPERPARAMETERS AND THE CORRESPONDING BINARY ENCODING DIGITS

Hyper-parameters	Choices	# Encoding Digits
# neurons	32, 64, 128, 256, 512, 768, 1024	3
# layers	1, 2, 3, 4	2
Activation functions	relu, elu, tanh, sigmoid	2
optimizers	rmsprop, adam, SGD, adagrad,adadelta, adamax, nadam	3

the offspring of the next generation will have substantial similarity between the compositions of each network's gene. To overcome this limitation, which might slow down the evolving process and keep the solution at a local optimal, distance calculation and force mutation play vital roles in ensuring that population can keep searching for more combinations in the later generations after identifying several network configurations with proper performance.

Genetic encoding (line 7) transforms one combination of four candidate hyperparameters into a unique binary string. Table I shows the available choices of each hyperparameter and the corresponding encoding bits. One-bit flipping (0 to 1) will result in changing of one hyperparameter, consequently affecting the performance of the network (e.g., for the choice of # layers, 00 to 10 means adding two more layers) . The Hamming distance calculation (line 9) is used to generate the distance matrices $H = (\sigma(x, y))$, where $1 \leq x \leq P$ and $1 \leq y \leq P$, and P is the designated number of individuals in one population. Lines 13 - 31 illustrate the procedure of all the genetic operations (conditional random selection, crossover, and force mutation) within specified activating conditions (defined in line 1). While doing the crossover operation, we select two network genes as the candidates of the parents. Each gene representing the combination of four hyperparameters is separated into two parts. Part A consists of the first two hyperparameters in the table, and part B takes the rest. By evenly separating one combination into two opposite parts, the proposed approach holds a lower bound p_s of the survival probabilities under simple crossover.

$$p_s = 1 - \delta(i,j)/(l-1), \tag{2}$$

where $\delta(i,j)$ is the distance between the two digits i and j we are observing, and l is the total length of the gene sequence (10 in this case thus $i,j \in [1,10]$). As the number of neurons in one layer can significantly affect the choice of the number of layers, grouping the choices of the first two hyperparameters onto one side of the cutting point will provide higher survival probabilities compared to separating them into two parts. As the mutation operator randomly activates after crossover, the survival probabilities reduced to

$$p_s = 1 - \delta(i,j)/(l-1) - (1-p_m)^{po}, \tag{3}$$

where p_m is the mutation probability. Instead of changing one bit of the genetic code, the mutation operator randomly chooses a value for one specific hyperparameter. Therefore, the maximum number of fixed position po is reduced from 10 to 4 (since we have 4 hyperparameters). By reducing po, the mutation effect increases ensuring a further decrease of the lower bound of the survival rate. We expected these changes could help on gene evolution process when most of the individuals in the later generations are very similar. Section IV further details how this strategy met our expectations. Again, note that this evolutionary process, while illustrative is not heuristic or strong and thus will not scale successfully in its present form.

B. Network Construction and Training Process

After evolving the networks for a certain number of generations, the last generation identifies the top candidate to construct the final training model. As the output layer separates the data into different classes, the number of neurons in the layer right before the output layer should not greatly exceed the number of classes for each specific task. Therefore, the number of neurons that the genetic selection process determines sets the number for only the first layer. The number of neurons will gradually diminish by half until either it reaches the last fully connected layer or the number of neurons in the current layer is less than twice that of the number of classes. Also, one 50 percent dropout layer is placed ahead of the output layer to reduce the effect of overfitting.

Moreover, we restricted the training epoch in the network selection phase to a relatively small but reasonable number (200), so the converging speed of the network has been considered by default. Nevertheless, the early stopping with patience=5 is set for all models, which means that most of the time the training process will not last until the last epoch. Ideally, we can identify a network that performs well in more generations during genetic selection within a competitive training duration.

IV. Experimental Results

We evaluate the proposed framework using two datasets; a disaster video dataset that consists of two major hurricane events that happened in 2017 in two geographic locations (Harvey in Texas and Irma in Florida), and a surveillance camera dataset that contains images captured from various places. Table II shows the statistical information of these two datasets. For the disaster dataset, by following chronological order, we use the first event as the training data while the later one becomes the testing data. We extract one keyframe image as the representative of each video. For the Network Camera 10K dataset, 20 percent of the data is separated into testing data. Moreover, 20 percent of the training data from both datasets was randomly selected to form the validation data for fitness score calculation which assists the model training and network evolution process.

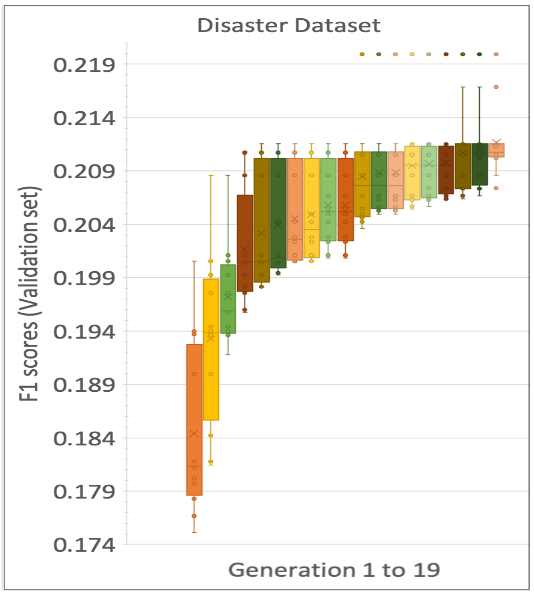

Figure 2. The performances of the 40 percent of individuals in each generation for the Disaster Dataset

Before getting into the evaluation of the final model, we observed the efficiency of the proposed framework. In Figure 2, the fitness scores' distribution (average F1 scores

Table II
THE STATISTICAL INFORMATION OF THE NETWORK CAMERA 10K AND DISASTER DATASET

Network Camera 10K						**Disaster**			
No.	**Concepts**	**Instances**	**No.**	**Concepts**	**Instances**	**No.**	**Concepts**	**Harvey**	**Irma**
1	Intersection	855	**8**	Yard	161	**1**	Demonstration	42	8
2	Sky	495	**9**	Forest	139	**2**	Emergency Response	81	20
3	Water Front	978	**10**	Street	431	**3**	Flood and Storm	426	177
4	Building+Street	603	**11**	Parking	99	**4**	Human Relief	70	1
5	Park	499	**12**	Building	243	**5**	Damage	42	172
6	Montain View	719	**13**	Highway	3724	**6**	Victim	75	16
7	City	432	**14**	Park+Building	149	**7**	Speak	347	63
Total			**9527**			**Total**		**1083**	**457**

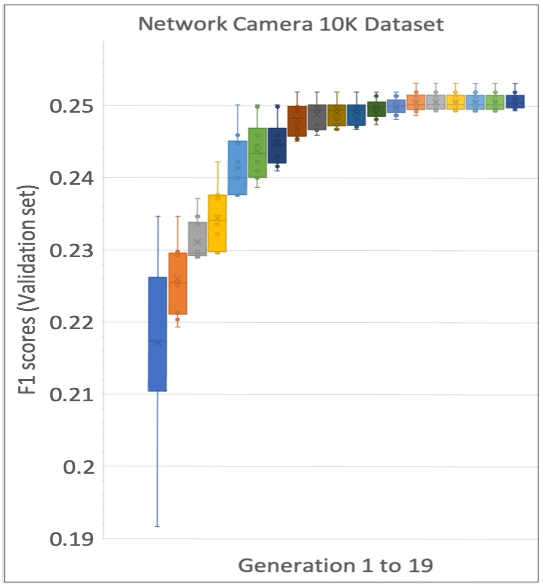

Figure 3. The performances of the 40 percent individuals in each generation for the Network Camera 10K Dataset

of the validation data) of the top 12 individuals in each generation depicts the evolutionary process. Since the retaining rate in the evolutionary process is defined as 0.4 and the number of populations in each generation is fixed at 30, only the top 12 individuals will survive and continue evolving in the next generation. As can be seen from the plot, the performance of each generation has steadily increased and reached a certain optimal F1 score near the 5th generation. After that, the new populations in each generation keep searching for a better solution and successfully exceed the optimal score in the 10th generation. Notably, the model not only focuses on discovering one individual as the best solution but also raises overall performance gradually for all the top populations in subsequent generations. Similar trends can be also found in Figure 3. Since we have a limited number of GPUs, we reduce the total number of populations in each generation to 25 for the Network Camera 10K dataset. As the retaining rate stays the same, the plot shows the performance of the top ten individuals for each generation. Those 10 populations in each generation are the parents that contribute to the next generation. Similarly, a local optimal appears near the 8th generation and sticks for several iterations. Again, our framework successfully gets the F1 score to improve after the 14th generation.

Furthermore, Figures 4 – 7 compare the search space covered in the representative generations and visualized all the individuals using scatterplots in three-dimensional space that clearly shows the improvement of the three generations (the first, the 10th, and the last generation). We project each unique network structure into two-dimensional space, where the x-axis represents the gene code in decimals of the first two hyperparameters, and the y-axis represents the other two. Five binary digits can be easily converted to a decimal number between 0 and 31. Therefore, a unique pair of x and y (point $[x, y]$ in the plots) represents a unique individual in the search space. The z-axis is the fitness score, which means the dots on the top represent the models that have better performance. The first generation starts with randomly selected individuals, covering a sparse space with various performance measures. Until the 10th generation, the individuals with lower scores are eliminated from the population. The solutions, however, are narrowed down into a smaller space. As we proposed a similarity sensitive framework, it still makes a breakthrough in the search space later on and produces better results. It is also evident in Figure 7 that the first generation (green dots) sparsely covers the search space by randomly producing 30 populations. Until the 10th generation (red dots), the individuals stay in a smaller solution area and demonstrate average performance. Finally, in the last generation (blue dots), the individuals became sparse again to cover a larger search space, which resulted in an optimal better than the local optimal than had been reached in the very early stage. We also plot the performance of the model using the Network Camera 10K dataset in the same way as shown in Figure 8. In the 10th generation, the network candidates have been identified in a very restricted area. Still, the proposed

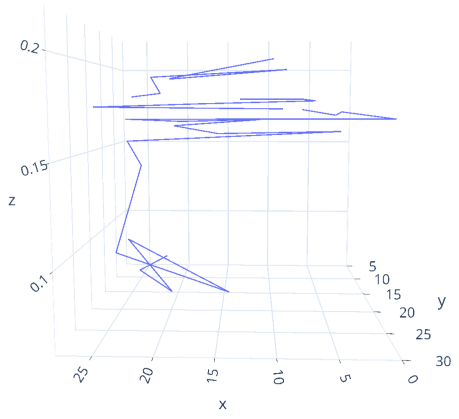

Figure 4. The first generation search

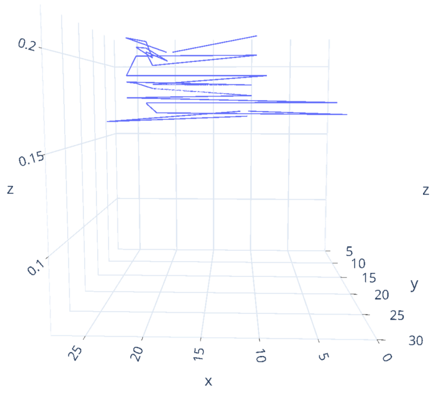

Figure 5. The tenth generation search

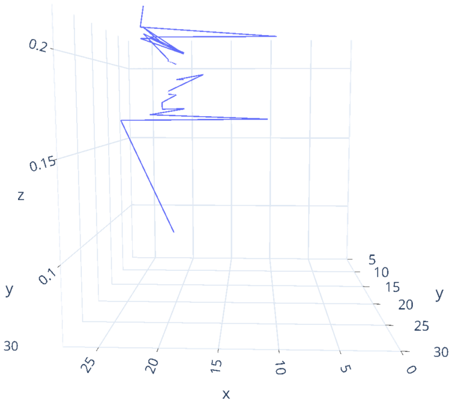

Figure 6. The last generation search

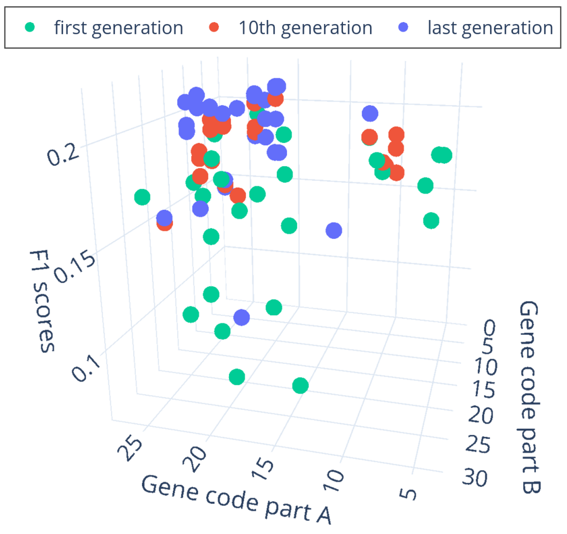

Figure 7. Individual performance in the first, tenth, and the last generation (Disaster)

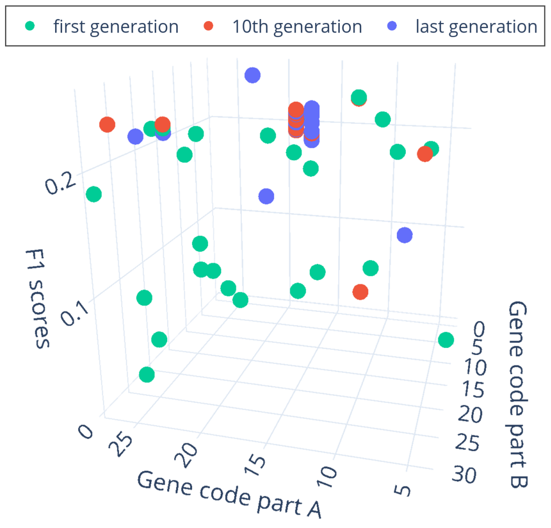

Figure 8. Individual performance in the first, tenth, and the last generation (Network Camera 10K)

framework shows the power of escaping the limited space and getting better solutions to construct the network with extensive performance.

Table III
EVALUATION RESULTS ON TWO DATASETS

Datasets	Models	AvgW. F1	Avg. F1
Disaster	MobileNet	0.380	0.121
	ResNet50	0.419	0.141
	Inceptin-v3	0.303	0.092
	Our Work	**0.541**	**0.194**
768 Neurons, 2 Layers, sigmoid, adamax			
Network Camera 10K	MobileNet	0.755	0.216
	ResNet50	0.773	0.233
	Inception-v3	0.726	0.194
	Our Work	**0.806**	**0.268**
256 Neurons, 1Layer, sigmoid, adamax			

The overall performance of the proposed framework is listed in Table III and compared with different pre-train models using two criteria (weighted average F1 score [AvgW. F1], and averaged F1 scores [Avg. F1], respectively). For evaluating the performance of an imbalanced dataset, F1 score is a more vital measure than accuracy. As trade-offs between precision and recall, F1 scores are more suitable to evaluate the overall model performance.

Generally, using a pre-trained CNN model for visual feature extraction and appending multiple network layers for image classification has proven able to get better results compared to constructing the model from scratch. Nevertheless, in this work, automated network construction performs much better than employing classical ML techniques to learn the high-level representations of the deep features. Results show increments of 12.2 percent and 5.3 percent respectively for weighted and unweighted average F1 scores for a new problem domain by adopting automated EA and considering the similarities between the solutions in the proposed work for the Disaster dataset. Compared to the pre-trained model using the Network Camera 10K dataset, the proposed method improved the weighted average F1 score by 3.3 percent. Seeing there is a significant improvement in F1 score for both datasets, we can conclude that the proposed framework selected and built the network model, which could recognize more instances correctly for each class. The table also shows the final configuration of the hyperparameters that achieved the scores as demonstrated.

V. Conclusions

In this work, we aim to leverage the deep features from pre-trained CNN models in different applications without spending most of the effort in examining the characteristics of each task. A generalized framework is proposed to accommodate all datasets. By integrating EA and other techniques to support the automated searching process, the proposed work determines the hyperparameters of a new neural network for one specific task after the best individual is selected. Overall, the experimental results have proven that a time-consuming task conducted by experts could be done by an automated process that surpasses human ability and reaches an optimal solution effectively. It should be noted however, that validation feedback needs to be provided lest the network select the best individual for an incorrect resultant task. Quadded neural nets could be applied to ameliorate this situation somewhat, but until the nets can be designed to reliably extract fundamental features and combinations of features, this possibility will persist. Again, the differential is attributed to "understanding" or the lack thereof. Higher-level evolution requires (self-referential) heuristics. An open question is how to represent and apply them in deep learning.

Acknowledgment

This research is partially supported by NSF CNS-1461926 and the Dissertation Year Fellowship (DYF) at Florida International University (FIU).

References

[1] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4694–4702.

[2] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, 2012.

[3] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[4] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[5] D. Wang and T. F. Zheng, "Transfer learning for speech and language processing," in *asia-pacific signal and information processing association annual summit and conference*. IEEE, 2015, pp. 1225–1237.

[6] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

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

[10] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[11] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *the 34th international conference on machine learning*, vol. 70. JMLR. org, 2017, pp. 2208–2217.

[12] H. Tian, S. Pouyanfar, J. Chen, S.-C. Chen, and S. S. Iyengar, "Automatic convolutional neural network selection for image classification using genetic algorithms," in *the IEEE international conference on information reuse and integration*. IEEE, 2018, pp. 444–451.

[13] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[14] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *the genetic and evolutionary computation conference*. ACM, 2017, pp. 497–504.

[15] H. Lam, S. Ling, F. H. Leung, and P. K.-S. Tam, "Tuning of the structure and parameters of neural network using an improved genetic algorithm," in *the 27th annual conference of the IEEE industrial electronics society*, vol. 1. IEEE, 2001, pp. 25–30.

[16] L. Xie and A. Yuille, "Genetic cnn," in *the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[17] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.