Genetic Algorithm based Deep Learning Model Selection for Visual Data Classification

Haiman Tian, Shu-Ching Chen
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
Email: {htian005, chens}@cs.fiu.edu

Mei-Ling Shyu
Department of Electrical and Computer Engineering
University of Miami
Coral Gabel, FL 33124, USA
Email: shyu@miami.edu

***Abstract*—Significant progress has been made by researchers in image classification mainly due to the accessibility of large-scale public visual datasets and powerful Convolutional Neural Network(CNN) models. Pre-trained CNN models can be utilized for learning comprehensive features from smaller training datasets, which support the transfer of knowledge from one source domain to different target domains. Currently, there are numerous frameworks to handle image classifications using transfer learning including preparing the preliminary features from the early layers of pre-trained CNN models, utilizing the mid-/high-level features, and fine-tuning the pre-trained CNN models to work for different targeting domains. In this work, we proposed to build a genetic algorithm-based deep learning model selection framework to address various detection challenges. This framework automates the process of identifying the most relevant and useful features generated by pre-trained models for different tasks. Each model differs in numerous ways depending on the number of layers.**

***Keywords*-transfer learning; deep learning model selection; genetic algorithms; image classification;**

I. Introduction

In this age of the Internet, people frequently interact with digital devices. These devices range from mobile phones, tablets, sensor-equipped infrastructures, vehicles, to smart household appliances. With this, we are experiencing a surge in data generation and transmission, which affects our everyday lives. More specifically, in multimedia data generation, which plays a vital role in both industrial applications and academic research [1][2][3][4]. This generation makes up 70% of the daily generated Internet data, and these vast amounts of data can be utilized to solve various domains' problems.

Advanced techniques, such as Deep Learning (DL), have been popularly used to investigate the different ways to take advantage of multimedia data analysis in different research fields [5]. Many astonishing research outcomes are generated with the assistance of DL approaches, including image classification [6], speech recognition [7], video understanding, etc. However, it is time-consuming and computationally expensive for each research group to build a DL model from scratch to fulfill their targeting solution. A method to better simulate a person's learning process is needed to generalize the knowledge to help solve these problems. Transfer learning, which provides the ability to transfer the experience from an original problem domain to a target domain, eases the learning process, and makes the well designed pre-trained models useful in a broader application domain as feature extractors [8][9].

Pre-trained deep learning models can extract different levels of features from the input data. However, for a variety of datasets, the feature strength is also varied [10]. Therefore, to efficiently identify the best model to generate the most representative features for the targeting problem domain is challenging. In the early stage, researchers always select the last layer before the prediction layer to extract the high-level abstract features for their specific tasks. It is uncertain whether this layer of every pre-trained model can always be the best choice considering the target domain is slightly different from the original problem domain. Hence, extracting features from other layers that carry lower level features might be more suitable for a particular target domain. Regarding examining a set of layers of each popular pre-trained model to obtain the best model for a specific task, we are looking for an optimal solution from a very large search space that exceeds the human ability. Therefore, an efficient and effective optimization/search algorithm is necessary to be used to automatically generate the feature set for a specific target problem. When the input dataset changes, the framework should have the ability to evaluate each model's performance regarding the new characteristics of the data, then select a new model/layer that generates the most representative feature to build the discriminative model.

To the best of our knowledge, there is no literature currently focusing on automatically determining the pre-trained deep learning model which fits a specific target domain. However, there are some optimization/search algorithms worth considering to tackle this problem. In this work, we proposed a Genetic Algorithm (GA) based deep learning model selection framework on identifying the feature set from a pre-trained model automatically. This feature set

contains the most representative features of a specific dataset that could potentially improve the model's performance. This generalized framework can accommodate different datasets and problem domains. By integrating a two-stage genetic code evolution process, the proposed approach identifies the best feature layer or the layers' combination for a specific task to further build an image classification model.

The rest of this paper is organized as follows. In section II, Convolutional Neural Networks (CNNs) and optimization/search algorithms for deep learning are discussed. 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

A. Convolutional Neural Networks

Essentially, Artificial Neural Networks (ANNs) are inspired by the behavior of different types of neurons in a biological system. A group of neurons that share the same properties will be responsible for the tasks related to a certain level, for example, detecting bright colors. The first level neurons' outputs will become a collection of inputs for the next level's neurons. ANNs can learn and recognize the observed patterns from this procedure.

Generally, CNNs are designed following a hierarchical architecture that consists of both linear and non-linear layers. Primarily, CNNs were intended to be utilized for basic image recognition, which made them standout amongst the most well-known and broadly utilized deep learning methods. Different from traditional Artificial Neural Network (ANN) models such as Multiple Layer Perceptrons (MLPs), which isolate the feature layers completely, CNN models take a raw image as input with a two-dimensional structure and share the feature weights among local neuron connections. This change significantly reduced the number of parameters and made the model simpler and easier to learn.

Many CNN models are built and trained on ImageNet, a large scale public image dataset, and can be utilized in transfer learning to tackle visual data classification tasks in a broader target domain. Inception V3 [11] is an updated version of GoogleNet, which have the convolutional and pooling layers separated in parallel. ResNet [12] overcomes the potential overfitting and vanishing gradient issue by constructing residual modules, which increase the depth of the model. MobileNet [13] is an efficient lightweight CNN model for mobile and embedded vision applications. The standard convolutions are factorized into pointwise convolutions and depthwise convolutions. DenseNet, proposed by Huang *et al.* in 2016 [14], connects every layer to every other layer in a feedforward fashion. This modification obtains significant improvement by strengthening the feature propagation and encouraging the reuse of feature, which substantially reduce the number of parameters.

B. Search and Optimization Algorithms

There has been an increase in demand for automated optimization technology across the various industries in the world of business and technology. The search algorithms have a high capacity when it comes to delivering better designs within a short period of time. Choosing the most efficient optimization/search algorithm for a particular problem is dependent on the already defined design space. Some of the available algorithms include genetic algorithm, evolutionary programming, grid search, random search, and Bayesian optimization. The availability of those algorithms has enhanced performance across a wide range of problems, which eliminates the need for manual tuning.

Both genetic algorithms and evolutionary programming are population-based optimization algorithms that incorporate many biological evolution operations to improve the quality of the solutions iteratively [15]. The operations include reproduction, mutation, recombination (a.k.a. crossover), and selection. A fitness function is defined to evaluate the health of each individual during the evolution process. Generally, a genetic algorithm is used to find precise solutions to both optimization and search problems, including finding either the minimum or the maximum function [16]. Compared to traditional methods, a genetic algorithm progresses from a population of candidate solutions, hence minimizing the chances of finding a local optimum. They can function under a noisy, nonlinear space, and are flexible to adjust. Recently, researchers have seen working on ways in which genetic algorithms can be used with evolutionary computation such as neural networks. Evolutionary programming is used in evolution simulation and to maximize the suitability of multiple solutions within an objective function. It relies on a known gradient within the search space when applied to design problems whose objective is the creation of new entities [17]. The recombination operation is eliminated from evolutionary programming because it considers each individual as an independent species. However, its advantage is the same as that of genetic algorithms, where no assumption is made about the underlying fitness landscape. Compared to other methods, they perform well on approximating solutions for nearly all types of problems and act efficiently when combined with neural networks.

Grid search is used to perform hyperparameter tuning to determine the optimal value for a specific model. Compared to genetic algorithms, grid search helps to find near-optimal parameter combination within specified ranges, such as support vector machine parameter optimization [18]. Gradient-based optimization can be applied to the optimization of neural network's learning rate separately for every iteration and layer. Compared to manual tuning, it enhances the ability to learn completely new data sets. However, the main disadvantage is that backpropagation across the entire training procedure requires a lot of time.

Random search algorithms are used to randomly select a representative samples from a given search space in order to identify the optimal value in the sampling [19][20]. It does not require derivatives to search in a continuous domain. Compared to grid search, the chances of finding optimal parameters are higher because of the random search pattern. Random search is faster than exhaustive search, but it is unreliable in determining the optimal solution.

A Bayesian optimization algorithm is a powerful tool when it comes to joint optimization design choices due to its ability to increase both product quality and productivity of human beings through an enhanced automation capacity [21]. It has been popularly used in many application domains, including interactive user-interfaces, environmental monitoring, automatic network architecture configuration, and reinforcement learning. Primarily in reinforcement learning, Bayesian optimization is used to tune the parameters of a neural network policy automatically, and to learn value functions at advanced levels of the reinforcement learning hierarchy. The technology can also be used to determine attention policies within image tracking with the use of deep neural networks. Compared to manual tuning methods, this approach can be used to tune many parameters simultaneously, which is essential for machine learning systems. The disadvantage with this technology, however, is that it is independent and relies on an optimizer to search the surrogate surface. Different from the general problem domains that we have observed, Bayesian optimization attains a superior performance, the relationship between each layer's feature performance for a specific CNN model is unknown. Since Bayesian optimization assumes that the solution space reflects the posterior probability distribution, it is uncertain if it is a good fit of Bayesian optimization for deep learning model selection.

III. Proposed Framework

Figure 1 illustrates the overall structure of the proposed automated model selection framework. The process for each candidate pre-trained deep learning model is independent and can be run in parallel. Therefore, the processing time of the framework will not be significantly affected when adding more models to the comparison. This framework consists of two genetic code evolution processes to determine the best feature set for a specific input dataset. First, the top feature layers from each candidate model are selected in the layer selection phase. For each model $(1, 2, ...N)$, the number of layers that we extract features from will change $(X, Y, ...K$ layers). Therefore, the genetic code generation accommodates the encoding of each feature layer as an individual with a fixed length considering the maximum number for each model. Then, the best feature combination is evaluated during the feature selection phase to generate the final features. This time, the encoding strategy changes to represent different combinations of the top layers. During the genetic code evolution process, several genetic operations are used to improve the average performance in each population. Each model's performance is validated in parallel using the best feature set. Only the model that shows the best performance on the validation data will be selected as the best model at the end.

A detailed explanation of the proposed framework is described next.

A. Genetic Code Evolution

Algorithm 1: Genetic Code Evolution

```
1  RETAIN ← 0.4
2  SELECT ← 0.1
3  MUTATE ← 0.2
4  for individual i ∈ Population p do
5      calculate FITNESS FUNCTION f(i)
6      grade[i].score ← f(i)
7  Sort grade in descending order
8  topGrade = grade[0 : RETAIN * grade.size]
9  restGrade = grade[RETAIN * grade.size :
     grade.size]
10 for x ∈ topGrade do
11     parents.append(x)
12 # Random selection
13 for x ∈ restGrade do
14     if SELECT > random() then
15         parents.append(x)
16 #Mutation
17 for x ∈ parents do
18     if MUTATE > random() then
19         MUTATE(x)
20 # Crossover
21 size ← Population.size − parents.size
22 while children.size < size do
23     select famale and male randomly from parents
24     if female ≠ male then
25         child = (male.partA + female.partB)
26         children.append(child)
27 parents.append(children)
28 return parents
```

Both the layer selection and feature selection phases use the same strategy to evolve the individuals, as shown in Algorithm 1. The initialization process randomly selects a certain number of individuals (we set it to 10 individuals empirically) and calculates the fitness score for each one of them. The fitness score is generated by the fitness function $f(i)$ (line 5), which is the average F1 score (Avg. F1):

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

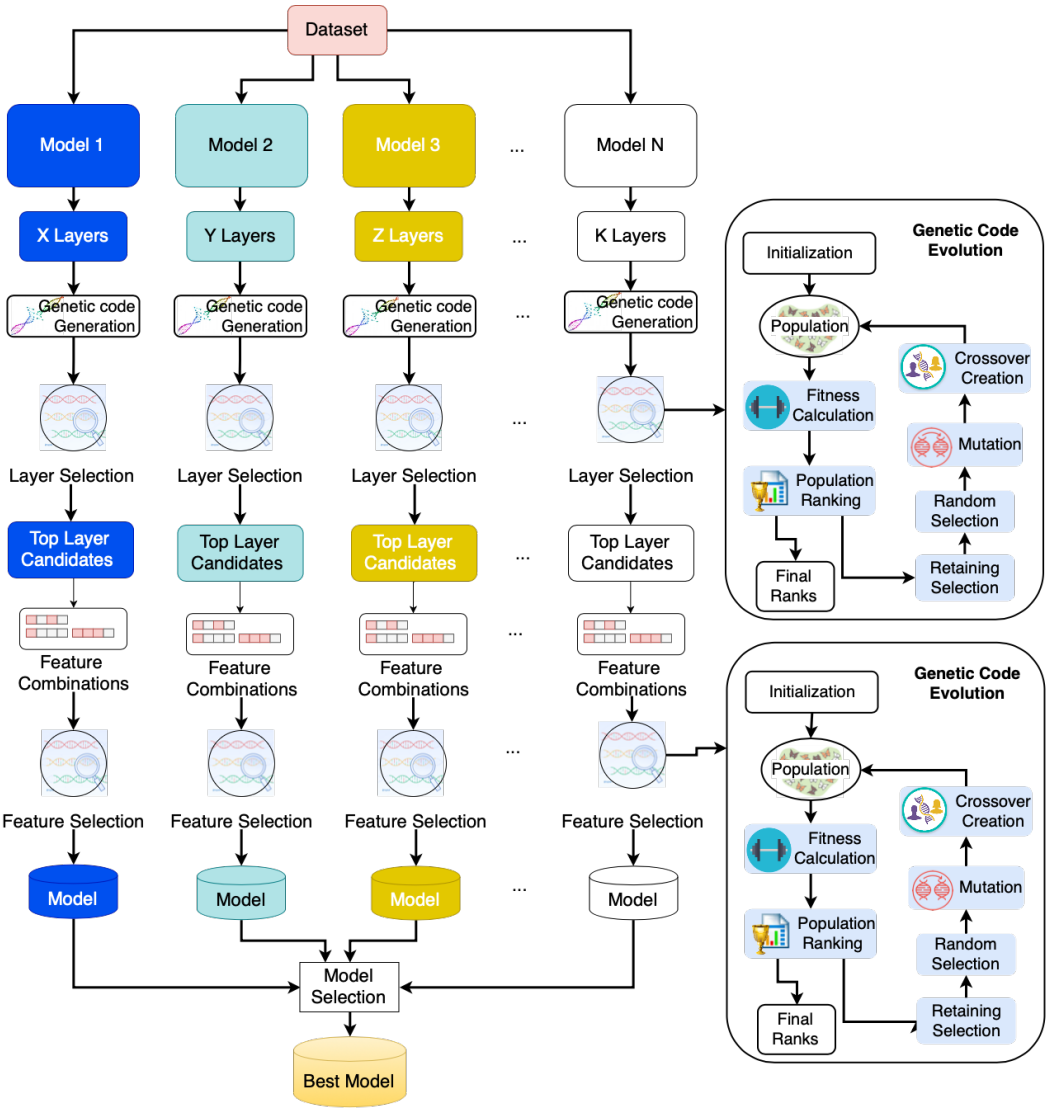

Figure 1. Proposed framework for deep learning model selection using a genetic algorithm

where C is the total number of classes in the target dataset; i is a unique individual; P is the precision of class c, and R is defined as the recall of class c. Precision represents the classifier's ability to not label a positive sample as negative, while recall represents the classifier's ability to find all the positive samples. The relative contribution of precision and recall to the F1 score is equal, which makes it a trade-off between these two evaluation criteria.

The individuals in a specific population are ranked in descending order to create a ranking list. Based on a predefined retention, the individuals on the top of the list will continue to the next generation and will produce offspring. Other individuals will have a small chance to survive depending on the random selection process in lines 12 - 15. All other individuals will be discarded for the next generation.

Figure 2 and 3 depict the genetic code evolution process for one generation for the layer selection and feature selection phases respectively. The figures illustrate the process of the mutation and crossover operations referring to lines 16 - 19 and lines 20 - 27 in the algorithm.

B. Layer Selection Phase

In the layer selection phase, genetic encoding operation transforms the ID of each feature layer into a unique binary string. The available layers for feature extraction in each model are different, which makes the corresponding encoding bits different for each model's process. For each individual, the features from a particular layer are extracted using the training data to build a Linear SVM classifier. Furthermore, the features from the same layer are obtained using the validation data to evaluate the classifier's performance and calculate the corresponding individual's fitness

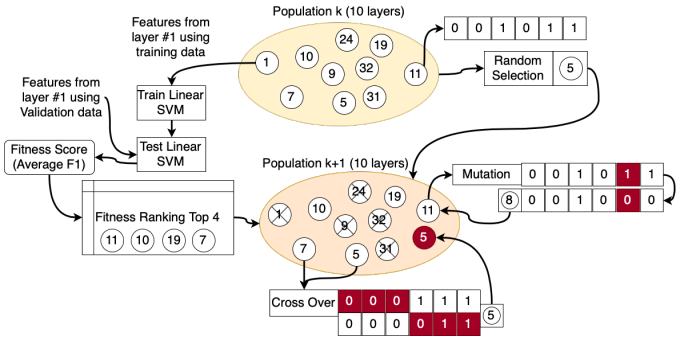

Figure 2. Genetic code evolution example for one generation in the layer selection phase

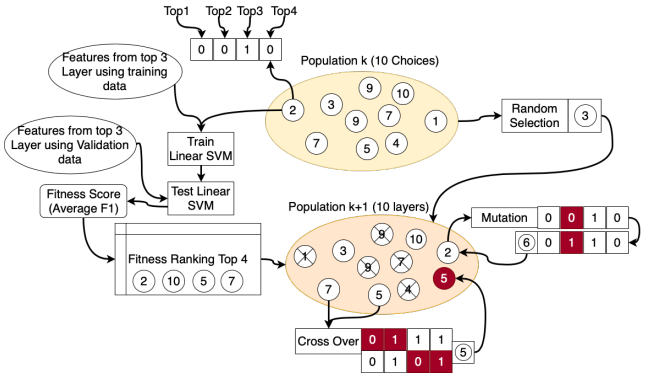

Figure 3. Genetic code evolution example for one generation in the feature selection phase

score.

A one-digit change (0 to 1 or 1 to 0) in the evolution process will result in choosing a different layer for feature extraction, affecting the performance of the classifier. For instance, changing 001011 to 001001 means that layer #8 will be selected instead of layer #11 to generate the features. This operation applies to the mutation process, where we restricted the process to affect only one position of the encoding each time for one selected individual.

The crossover operation generates new individuals for the next population by combining the genetic codes from two retained individuals. Each parent contributes only the left half or the right half of the genes. A new individual is then added to the next population by combining these two halves to create a new genetic code of the same length. For example, taking the left half of the genetic code 000111 and the right half of the genetic code 000011 will generate another individual represented by the genetic code 000011.

C. Feature Selection Phase

After evolving the individuals in the layer selection phase for several generations, the last generation identifies the top layer candidates to represent the most reliable features for a specific dataset. The best individuals are determined by the predefined retention rate from the final ranking list.

Table I
THE PRE-TRAINED DEEP LEARNING MODEL CANDIDATES WITH THE AVAILABLE NUMBER OF FEATURE CHOICES

Models	Layers	# combinations
InceptionV3	94	94^4
ResNet50	64	64^4
MobileNet	13	13^4
DenseNet201	80	80^4
Total combinations	–	$3.41E32$

Those features will be further encoded as different feature combinations to proceed with feature selection.

Different from the previous stage, where each genetic code represents a single feature layer, in the feature selection phase each binary string encodes a way of combining features from different layer candidates. The mutation and crossover operations as described in the previous section remain the same, except that each digit means a top feature set will be selected or deselected to form the final feature set (e.g., a "0" means do not select, while a "1" means select). Therefore, a mutation process will either add a new feature layer or remove a feature layer from the final feature set.

After finishing the second phase of the genetic code evolution process, we selected the top feature set from each pre-trained deep learning model with the highest average F1 score running on a Linear SVM classifier. The final model is determined by comparing the average F1 scores using the same validation data to extract features from each model. The model with the highest score will be selected as the best feature extractor to build the final classification model.

IV. Experimental Analysis

A. Experimental Setup

We chose four pre-trained deep learning models as our model candidates. The model and the corresponding number of available layers are shown in Table I. As we set each population to generate 10 individuals, and the retention rate (r) to 0.4, for each model's layer selection phase we have a total number of K^{10*r} feature set choices, where K is the available number of layers for each model. As the final output is limited to the feature set from one model, the total number of possible choices to determine an optimal solution adds up to $3.41E32$. The space is far too large to be explored exhaustively by hand.

We used four datasets from different domains to evaluate our proposed approach: two imbalanced and two balanced datasets. One of the imbalanced datasets is a disaster video dataset that consists of two major hurricane events that happened in 2017 in two different geographic locations: Harvey in Texas and Irma in Florida. The other imbalanced dataset is a surveillance camera dataset that contains images captured from a variety of places. Table II shows the statistical information of these two datasets. In the Disaster dataset, the "Flood and Storm" concept contains most of the samples.

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**

For the disaster dataset, by following a chronological order, we use the first event as the training data and the second event as the testing data. We extracted one representative keyframe image for each video. For the Network Camera 10K dataset, 20 percent of the data was separated into testing data. Moreover, 20 percent of the training data from both datasets was randomly selected to form the validation data for the fitness score calculation. The majority class in this dataset is the concept "Highway". The two balanced datasets (CIFAR10 and MNIST-Fashion) are well-known public datasets. CIFAR10 classifies objects and animals, and MNIST-Fashion serves as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. These two datasets were already split into training and testing, but we randomly selected 20% of the training samples for our validation data to calculate the fitness score during the genetic code evolution process. Both datasets consist of an equal number of samples for each class. CIFAR10 includes concepts related to objects (e.g., airplane, automobile, ship, and truck) and animals (e.g., bird, cat, deer, dog, frog, and horse). MNIST-Fashion is a collection of grayscale images of clothing types such as t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

We compared the performance of our proposed framework with that of the three optimization algorithms mentioned in the related work. Each of those algorithms selects a pre-trained model as the best feature model. Bayesian Optimization is chosen to evaluate if its advantage regarding probability assumptions will have a positive impact on our specific task. Evolution programming and genetic algorithm without mutation operations are included in the comparison to determine whether or not both mutation and crossover operations are necessary to converge to the optimal solution.

B. Experimental Results

The performance of the proposed framework compared to the other three optimization algorithms on the four datasets and with different pre-trained models is shown in Table III. Four metrics were considered: Precision, Recall, averaged F1 scores [Avg. F1], and Weighted average F1 score [W. Avg. F1]. For evaluating the performance of an imbalanced dataset, the F1 score is a a better measure than accuracy. As the F1 score captures the trade-off between precision and recall, it is more suitable to evaluate the overall model performance. In this framework, we use a Linear SVM model to evaluate the feature performance on the validation data. Therefore, the reported evaluation metrics are based on the testing results of the SVM classifier's output.

As shown in Table III, our proposed framework always selected the pre-trained model with the best performance on each dataset. For the disaster dataset, only our proposed method selected ResNet50 to extract the feature set. The performance improved more than 5% compared to the others using W. Avg. F1. For Network Camera 10K dataset, though Evolutionary Programming selected the same model as our proposed method, the overall performance is the worst, which means it failed to identify the best feature set. The other two models selected ResNet50 to produce the final feature set, while our method select InceptionV3 and has a better performance on the testing.

For the balanced public datasets, CIFAR 10 and MNIST-Fashion, all methods selected ResNet50 as the best feature model. We didn't report W. Avg. F1 scores here because they are identical to the Avg. F1 scores when each class has the same number of samples. Though all the methods selected the same pre-trained model, the features performances are not all the same. Our method is the only one can bring CIFAR 10 data's performance beyond 90%. Though all the methods' performance are very close in MNIST-Fashion, our method still identifies the feature set with the best performance.

Figure 4 - 7 illustrate the single model's feature performance using the proposed model and the other three optimization algorithms. The y-axis in all the figures represents the evaluation metrics' scores (ranging between 0 to 1). The purpose of these comparisons here is to ensure that our proposed method could always determine the best feature set for a specific dataset no matter how the candidate models change. Though DenseNet201 and MobilenNet models are not selected by any one of the optimization algorithms for the four experimental datasets, from the bar chart we can tell, the proposed method always have the best performance consider all the evaluation metrics. Figure 7 also shows that

Table III
PROPOSED FRAMEWORK'S FINAL MODEL PERFORMANCE ON FOUR DATASETS COMPARE TO BAYESIAN OPTIMIZATION, EVOLUTIONARY PROGRAMMING, AND GENETIC ALGORITHM WITHOUT MUTATION OPERATION

Datasets	Algorithms	Final Model	Precision	Recall	Avg. F1	W. Avg. F1
Disaster	Bayesian Optimization	InceptionV3	0.3215	0.3256	0.2747	0.3920
	Evolutionary Programming	InceptionV3	0.3192	0.3084	0.2514	0.3937
	Genetic Algorithm w/o mutation	InceptionV3	0.3215	0.3256	0.2747	0.3920
	Proposed Method	**ResNet50**	0.3212	**0.3276**	**0.2867**	**0.4430**
Network Camera 10K	Bayesian Optimization	ResNet50	0.6398	0.6263	0.6290	0.7827
	Evolutionary Programming	InceptionV3	0.6644	0.5896	0.6108	0.7705
	Genetic Algorithm w/o mutation	ResNet50	0.6391	0.6081	0.6175	0.7761
	Proposed Method	**InceptionV3**	0.6508	**0.6339**	**0.6409**	**0.7985**
CIFAR10	Bayesian Optimization	ResNet50	0.8949	0.8943	0.8945	-
	Evolutionary Programming	ResNet50	0.8996	0.8995	0.8995	-
	Genetic Algorithm w/o mutation	ResNet50	0.8934	0.8928	0.8930	-
	Proposed Method	**ResNet50**	**0.9063**	**0.9061**	**0.9061**	-
MNIST -Fashion	Bayesian Optimization	ResNet50	0.9260	0.9263	0.9260	-
	Evolutionary Programming	ResNet50	0.9282	0.9285	0.9282	-
	Genetic Algorithm w/o mutation	ResNet50	0.9282	0.9285	0.9282	-
	Proposed Method	**ResNet50**	**0.9289**	**0.9292**	**0.9289**	-

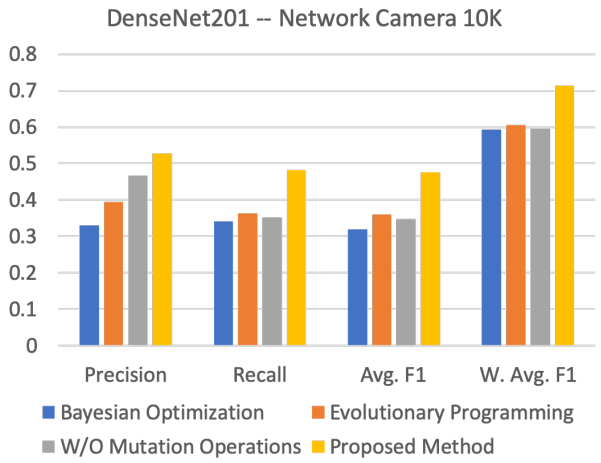

Figure 4. DenseNet201 model performance on Network Camera 10K dataset

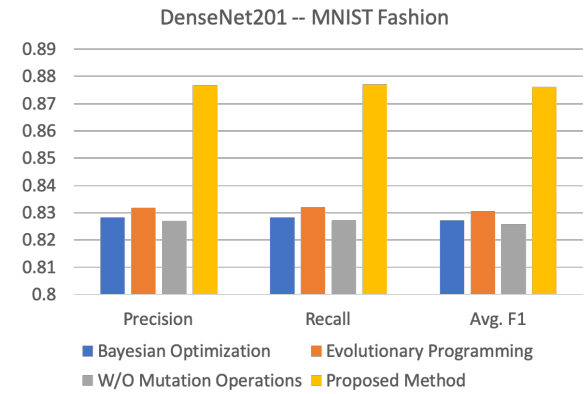

Figure 5. DenseNet201 model performance on MNIST-Fasion dataset

the proposed method identifies the best feature set from InceptionV3 model, however it does not select it as the

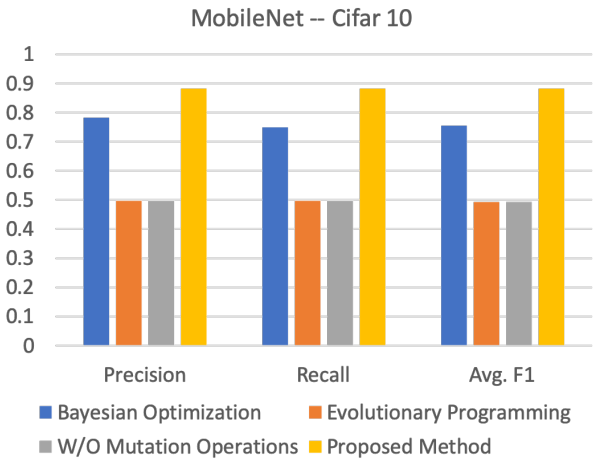

Figure 6. MobileNet model performance on CIFAR10 dataset

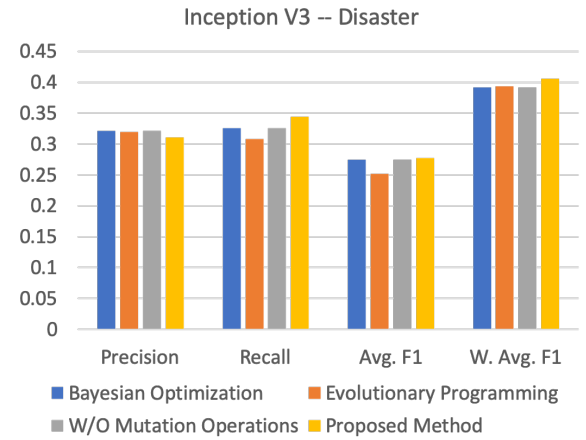

Figure 7. InceptionV3 model performance on Disaster Dataset

best model. Thus, it is true that this model's best feature set's performance cannot compete with the feature set from ResNet50 model as we showed in the experimental result table.

V. Conclusion

We identify the potential challenges of using pre-trained deep learning models on different target problem domains. We proposed to build a generalized framework using genetic algorithms to automatically determine the best feature set from a group of model candidates. A feature set that contains the most representative features for a specific target domain can be better utilized to train a classifier, then further enhance the final model's performance. The experimental results have shown that our proposed approach outperformed the other optimization algorithms and can always identify the best feature set no matter how the model candidates change. Since each model candidate is processed and evaluated independently, the framework can be run in parallel and makes the time-consuming task to be more efficient.

Acknowledgment

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

References

[1] S. Pouyanfar, Y. Yang, S.-C. Chen, M.-L. Shyu, and S. Iyengar, "Multimedia big data analytics: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, p. 10, 2018.

[2] W. Zhu, P. Cui, Z. Wang, and G. Hua, "Multimedia big data computing," *IEEE multimedia*, vol. 22, no. 3, pp. 96–c3, 2015.

[3] X. Li, S.-C. Chen, M.-L. Shyu, and B. Furht, "Image retrieval by color, texture, and spatial information," in *Proceedings of the 8th international conference on distributed multimedia systems*, 2002, pp. 152–159.

[4] S.-C. Chen, M.-L. Shyu, and R. Kashyap, "Augmented transition network as a semantic model for video data," *International Journal of Networking and Information Systems*, vol. 3, no. 3, pp. 9–25, 1998.

[5] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, p. 92, 2018.

[6] 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 *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 2018, pp. 444–451.

[7] D. Wang and T. F. Zheng, "Transfer learning for speech and language processing," in *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE, 2015, pp. 1225–1237.

[8] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *arXiv preprint arXiv:1611.06440*, vol. 3, 2016.

[9] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[10] 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.

[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[14] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[15] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[16] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," in *Feature extraction, construction and selection*. Springer, 1998, pp. 117–136.

[17] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary computation*, vol. 3, no. 2, pp. 82–102, 1999.

[18] P. Lameski, E. Zdravevski, R. Mingov, and A. Kulakov, "Svm parameter tuning with grid search and its impact on reduction of model over-fitting," in *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*. Springer, 2015, pp. 464–474.

[19] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," *arXiv preprint arXiv:1803.07055*, 2018.

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

[21] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.