Information Systems Frontiers (2020) 22:1053–1066
https://doi.org/10.1007/s10796-020-10023-6

Evolutionary Programming Based Deep Learning Feature Selection and Network Construction for Visual Data Classification

Haiman Tian[1] · Shu-Ching Chen[1] · Mei-Ling Shyu[2]

Published online: 25 June 2020

Abstract
Convolutional Neural Network (CNN) models and many accessible large-scale public visual datasets have brought lots of research work to a remarkable new stage. Benefited from well-trained CNN models, small training datasets can learn comprehensive features by utilizing the preliminary features from transfer learning. However, the performance is not guaranteed when taking these features to construct a new model, as the differences always exist between the source and target domains. In this paper, we propose to build an Evolution Programming-based framework to address various challenges. This framework automates both the feature learning and model building processes. It first identifies the most valuable features from pre-trained models and then constructs a suitable model to understand the characteristic features for different tasks. Each model differs in numerous ways. Overall, the experimental results effectively reach optimal solutions, demonstrating that a time-consuming task could also be conducted by an automated process that exceeds the human ability.

Keywords Deep learning · Evolutionary programming · Image classification

1 Introduction

The Internet era keeps people connected through all kinds of digital devices. For example, interactions happen during the active use of mobile phones, tablets, the Internet of Things (IoT) devices, vehicles, and smart household appliances. These devices are all connected and can further affect our everyday lives by generating and conveying waves of data in one second (Chang 2019; Mukherjee 2020). Along with this data generation, multimedia data (Chen et al. 1998, 2001, 2005), as a vital part that makes up 70% of the daily Internet traffic, has been utilized frequently to solve problems at various domains for both industrial applications and academic research (Li et al. 2002; Pouyanfar et al. 2018; Zhu et al. 2011, 2015).

Various advanced techniques are popularly used to take the full advantage of these multimedia data in different research fields (Chen et al. 2013; Chen and Kashyap 2001; Chen et al. 2006; Lin and Shyu 2010). Among these, Deep Learning (DL) approaches (Pouyanfar et al. 2018) have generated many astonishing research outcomes in different areas, such as multimedia research including image classification (Tian et al. 2018), speech recognition (Wang and Zheng 2015), video understanding, etc. However, DL approaches are usually time-consuming and computationally expensive. Hence, a DL model built from scratch for an individual research group's target problem is not always accessible. Most researches benefit from much more generalized knowledge that better simulates a person's learning process on solving certain kinds of problems, namely transfer learning (TL) (Pouyanfar et al. 2019; Tian et al. 2019). TL can transfer the knowledge obtained from one problem domain to another related target domain. It eases the initial learning process by utilizing the well designed and complicated pre-trained models as feature extractors (Molchanov et al. 2016; Shin et al. 2016).

A pre-trained deep learning model can be seen as a collection of feature extractors, which extract different levels of

✉ Haiman Tian
htian005@cs.fiu.edu

Shu-Ching Chen
chens@cs.fiu.edu

Mei-Ling Shyu
shyu@miami.edu

1 School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA

2 Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL 33124, USA

features from one input source. As the input datasets are different, the feature strength is not always stable (Pan and Yang 2010; Tian et al. 2018). Thus, identifying a model that can present the most competent feature set for the target domain is still challenging. Previously, the last activation layer, which directly connects to the prediction outputs, is always considered as the best layer to extract the high-level features that are suitable for most of the tasks, regardless whether this layer in each pre-trained model always produces the most representative features for every target domain. Furthermore, differences between the target domain and the original problem domain still exist. Thus, it is worth considering extracting lower level features from other layers for a particular target domain. However, finding the best layers from a group of popularly picked pre-trained models for a particular task creates a vast search space, which exceeds the human ability. Therefore, an efficient and effective optimization/search algorithm is required to optimize both the feature generation and feature learning processes for a specific target problem automatically. As the input data source changes, the framework should re-evaluate each pre-trained model's feature strength and build a new model with the most representative feature layers to catch the new characteristics of the data (Tian et al. 2019).

There is still very little research work in the literature currently focusing on automatically determining the pre-trained deep learning model that fits a specific target domain (Tian et al. 2018, 2019). However, there are some optimization/search algorithms worth considering to tackle this problem. In this paper, we propose an Evolutionary Programming (EP) based DL feature selection and network construction framework that can identify the feature set from a pre-trained model and further construct a new feature learning model automatically. The new feature set contains the most representative features of a specific dataset that could potentially improve the model's performance, and the new network model takes the new feature set at the input. This generalized framework can accommodate different datasets and problem domains. By integrating the genetic code evolution process, the proposed approach identifies the best feature layer or the layers' combination for a specific task and further builds an image classification model. The hyperparameters of the new neural network are determined automatically after the best feature set is selected for each specific task.

The remainder of this paper is organized as follows. In Section 2, we review the literature and some existing work in Convolutional Neural Networks (CNNs) and optimization/search algorithms for deep learning. Section 3 presents the proposed framework and explains each component in details. The datasets and experimental results are discussed in Section 4. Finally, we conclude the paper in Section 5.

2 Related Work

2.1 Convolutional Neural Networks

Inspired by the biological system, Artificial Neural Networks (ANNs) mimic the behavior of different types of neurons. Neurons with the same properties are responsible for completing a portion of a complex task at a particular level, for example, detecting bright colors. In each level, all neurons respond to the input signals and output new signals to the next level which form as new inputs for other types of neurons. From this iterative procedure, ANNs learn and recognize the characteristic patterns (Tian and Chen 2017).

Different from traditional ANN models, such as Multiple Layer Perceptrons (MLPs), which consider each neuron as independent in the feature layers, CNN models take a raw image as the input and share the feature weights among local neuron connections within a two-dimensional structure. This change significantly reduced the total number of parameters of the model and made the learning process more straightforward. Essentially, CNNs standout amongst the most well-known and broadly utilized deep learning methods for image recognition. A typical design of the CNN model follows a hierarchical architecture that consists of both linear and non-linear layers.

ImageNet is a large scale image dataset, which is open to the public and popularly used to build and train the CNN models. By learning the feature distributions from this dataset, many models have obtained astonishing performance, even surpassing human identification capabilities, such as Inception V3, ResNet, DenseNet, and MobileNet. However, training a deep neural network has been proven to be NP-hard. Network models aim to find a local optimum that is good enough to approximate the global optimum. Inception V3 (Szegedy et al. 2016) can be seen as an upgrade from GoogleNet, which places the convolutional and pooling layers in parallel. ResNet (He et al. 2016) introduces the residual model to overcome the overfitting and vanishing gradient issues that will potentially happen while increasing the depth of the model. DenseNet, proposed by Huang et al. (2017), connects every layer to every other layer in a feedforward way. This change obtains a notable enhancement by supporting the feature propagation and the reuse of features, which extensively reduce the number of parameters. MobileNet (Howard et al. 2017) is an efficient and lightweight CNN model, suitable for mobile and embedded vision applications. The regular convolution operations are factorized into two types of convolutions, namely pointwise convolutions and depthwise convolutions. All these models have been trained on very high-performance machines to learn the features that can represent the same set of image categories available in

ImageNet. Therefore, identifying the image concepts that were already defined in ImageNet can be done by utilizing those pre-trained models. Nevertheless, extracting the high-level features from an image that contains an unknown concept will always let the models classify it into the wrong categories.

Using version spaces to represent tentative heuristics enables the general models to keep track of all the useful information from a sequence of learning examples. Multiple models are managed within one version space to accomplish a concept learning process (Mitchell 1997; Winston 1992). Essentially, a version space represents all the plausible descriptions of a heuristic by incorporating both general and specific models. Tentative heuristics advance the way of feature learning that finds the instances known in both the generalized model and the specific model. It recognizes the absent relationships between the models without remembering the examples. Via TL, the pre-trained general models are further used to undertake visual data classification tasks in diverse target domains, which forms knowledge previously only known by the specific models. However, an automated approach to identify the best deep feature set and build a new model suitable for every new target is not available.

2.2 Search and Optimization Algorithms

There is an increasing demand for automated optimization across various industries. Search algorithms are highly capable of delivering better designs in a shorter time. Determining the most efficient optimization/search algorithm for a particular problem is dependent on the specified design space. Some potential algorithms are Genetic Algorithm (GA), Evolutionary Programming (EP), grid search, random search, and Bayesian Optimization (BO). Those algorithms have been applied to a wide range of problem domains to enhance the model performance, which eliminates the need for manual and exhausting parameter tuning.

Population-based optimization algorithms cover both GA and EP, which leverage various biological evolution operations to discover more powerful solutions iteratively (Back 1996). Essential operations are reproduction, selection, recombination (a.k.a. crossover), and mutation. An individual's wellness is evaluated during the evolving process by calculating a pre-defined fitness function. Consequently, the fitness score affects the decision of next operations on each individual. Usually, GA is used to find precise solutions, either the minimum or maximum function, for both optimization and search problems (Yang and Honavar 1998). Compared to traditional methods, GA and EP progress from a population of candidate solutions, can thus function under a noisy and nonlinear space, and are flexible to adjust. Recently, GA has started to be integrated with evolutionary computation in building neural networks. EP is used in ways of simulating the evolution 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 (Yao et al. 1999). The recombination operation is eliminated from EP as it treats each individual as an independent species. However, it holds the same advantage as GA, 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 take actions efficiently when integrated with neural networks. Evolutionary algorithms have also been utilized to provide feature selection functionalities in computer vision and machine learning. In Li et al. (2019), the authors introduced a Dividing-based Many-objective Evolutionary Algorithm for large-scale Feature Selection (DMEA-FS) which aims to satisfy both feature dimension reduction and stable classification accuracy. However, their approach is still limited to process a small number of manufactured features (less than 1000) with a small group of instances (less than 2000). DL models can quickly generate thousands of features in one layer that exceeds the total number of features their algorithm selects. Therefore, treating each feature as independent is not a reasonable approach. In addition, accuracy is not an appropriate performance metric for imbalanced datasets where some of the categories have very few instances.

Grid search is popularly used in machine learning model training, which performs hyperparameter tuning to determine the optimal value for a specific model. Compared to GA and EP, the ability of grid search is limited as it assists in finding near-optimal parameter combinations within specified ranges, for example, parameter optimization for Support Vector Machine (SVM) (Lameski et al. 2015). Gradient-based optimization can be applied to optimize a neural network's learning rate for every iteration and layer. Compared to manual tuning, it enhances the ability to investigate new datasets. However, its main disadvantage is that back-propagation consumes too much time to complete the entire training procedure. Random search algorithms randomly select representative samples from a given search space to identify the optimal value (Bergstra and Bengio 2012; Mania et al. 2018). It does not require derivatives to search in a continuous domain. Because of the pattern of random search, grid search algorithms have a higher chance of finding optimal parameters. Random search is faster than an exhaustive search, but it is unreliable in determining the optimal solution.

BO algorithms have been successfully applied in many research problem domains, such as interactive user-interfaces, environmental monitoring, automatic network architecture configuration, and reinforcement learning. Particularly in reinforcement learning, BO algorithms learn the value functions at advanced levels and tune the neural network policy's parameters automatically. It is a powerful tool that enhances the automation capacity for joint optimization design choices, as its ability increases both the product quality and productivity of human beings (Snoek et al. 2012). Besides reinforcement learning, BO algorithms can also learn the attention policies in image tracking using deep neural networks. Compared to manual tuning methods, the automatic approach can tune many parameters simultaneously, which is essential for machine learning systems. However, the BO algorithms are not independent and rely on an optimizer to search the surrogate surface, which becomes the algorithm's disadvantage. Since BO

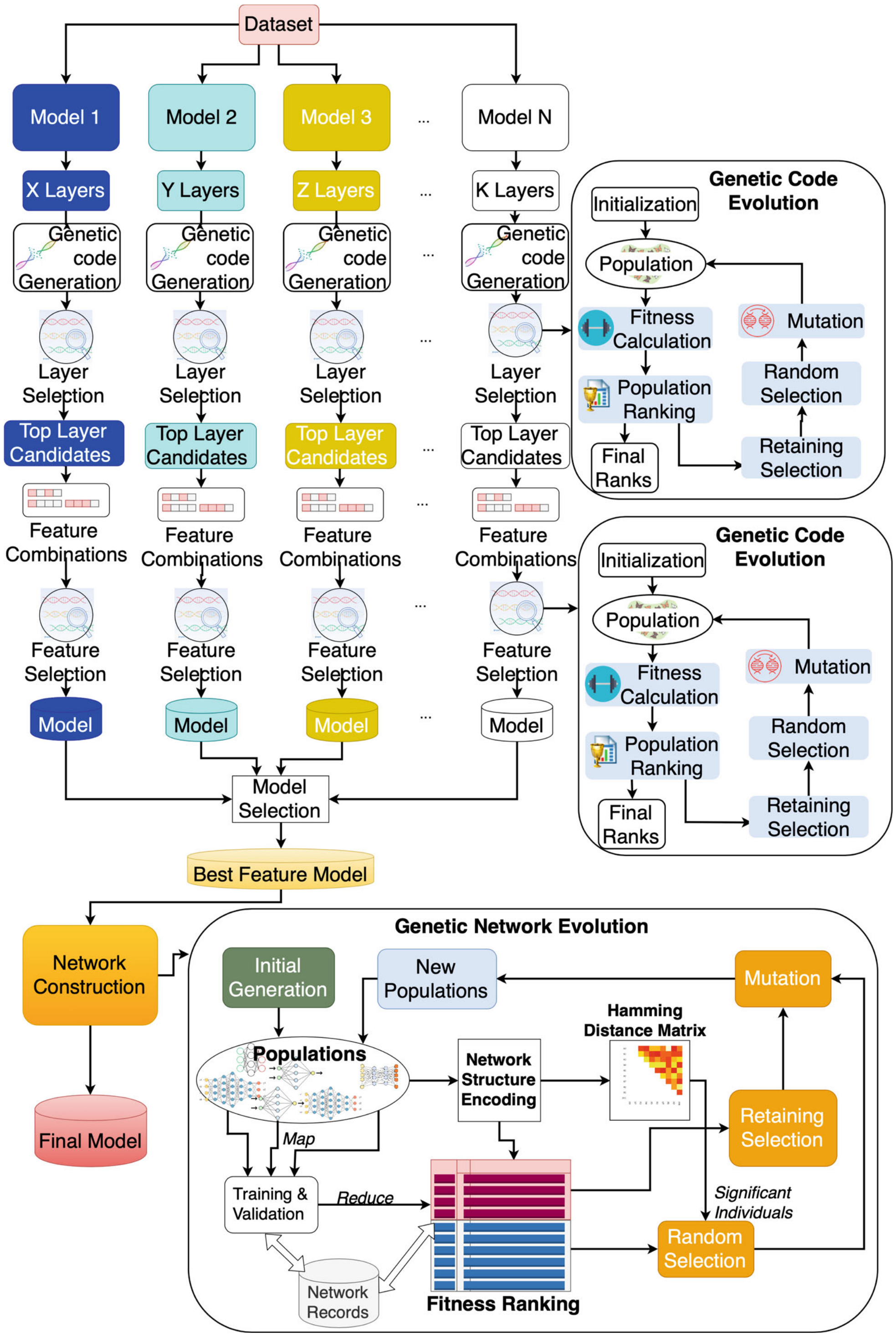

Fig. 1 Proposed framework for deep learning feature selection and network construction using evolutionary programming

algorithms assume that the solution space reflects the posterior probability distribution, it is not clear if BO algorithms are the right choice for deep learning model selection. Different from the general problem domains in which BO algorithms have attained superior performances, the correlation between each layer's feature strength for a specific pre-trained CNN model is unknown.

3 Proposed Framework

Figure 1 shows the structure of the proposed framework, which automatically selects the deep features from each candidate pre-trained deep learning model and then builds a new neural network to learn the new relationship between the features and the target concepts. As the feature sets from each pre-trained model are independent, the first stage of feature selection can then be run in parallel. Therefore, adding more candidacy models to the framework will not significantly increase the processing time for feature evaluation. This framework consists of two genetic code evolution processes to determine the best feature set for a specific input dataset and a genetic network evolution process to identify the best network structure that learns the connection between the representative features and the final concepts.

First, the layer selection phase selects top feature layers from each candidate pre-trained model (Model 1, 2, ..., N). Each model contains a different number of layers (X, Y, ..., K Layers) that can extract various levels of features. Genetic code generation encodes every feature layer as an individual with a fixed-length binary string, considering the maximum number of layers available in each model.

Then, the feature selection phase identifies the best feature combination via the same genetic code evolution procedure to generate the final feature set. The encoding strategy slightly changes to represent different options of the candidate feature layers. Genetic operations, such as retaining, random selection, and mutation, are leveraged in the genetic code evolution process to improve the individual's performance in each population. Each pre-trained model's performance is validated in parallel using the identified best feature set. Only the model that shows the best performance on the validation data can be chosen as the best feature model and be prepared for the final network model building.

In the network evolution process, each individual formulates a network's gene with four major hyperparameters (i.e., the number of neurons in one layer, the number of fully connected layers in one model, the activation function, and the optimizer). Every network's input is a set of features extracted from an individual layer or several layers of a pre-trained DL model. The general feature model and the representative feature layers have already been determined in the layer selection and feature selection phase. Therefore, it is not necessary to build another feature model that manipulates those features. Only a subset of the layers that have been identified to generate significant features from a complex pre-trained DL model is maintained. Those layers vary while the target problem domain changes. Then, a simpler model with a smaller number of hidden layers is constructed here to further strengthen the feature representations. The main idea here is not to entirely replace a DL model that contains more layers with a much simpler neural network, but to utilize the valuable feature layers from the well-trained DL model and then transfer the feature representations with a smaller neural network to better fit into the target problem domain. The genetic network evolution takes the same genetic operations as in the previous processes. Meanwhile, a Hamming distance matrix is calculated for every generation to evaluate the structural variations between individuals. Genetic operations respond to this similarity evaluation process, which ensures that the development of the new generation continues to explore the searching space. The network that obtains the best fitness score on the validation data is then identified at the end. Next, a complete training process starts to build the final model for the specific classification task.

Algorithm 1 Genetic code evolution.

```
$RETAIN \leftarrow 0.4$; $SELECT \leftarrow 0.1$; $MUTATE \leftarrow 0.5$
GeneEncoding();
$Population \leftarrow$ Initialization()
for $individual\ i \in Population\ p$ do
    calculate FITNESS FUNCTION $f(i)$
    $grade[i].score \leftarrow f(i)$
Sort $grade$ in descending order
# Population Retaining
$topGrade = grade[0 : RETAIN * grade.size]$
$restGrade = grade[RETAIN * grade.size : grade.size]$
for $x \in topGrade$ do
    $parents.append(x)$
# Random selection
for $x \in restGrade$ do
    if $SELECT > random()$ then
        $parents.append(x)$
$size \leftarrow Population.size - parents.size$
# Mutation
while $children.size < size$ do
    select $candidate$ randomly from $parents$
    for $i \in range(candidate.length)$ do
        if $MUTATE > random()$ then
            $MUTATE(candidate[i])$
    $children.append(candidate)$
$parents.append(children)$
return $parents$
```

3.1 Genetic Code Evolution

Algorithm 1 illustrates the genetic evolution process used in both layer selection and feature selection phase. $GeneEncoding()$ function sets up the restriction for the initialization process that handles the differences between these two scenarios. For example, layer selection converts each particular layer ID into a fixed length binary string. The length of the gene string is fixed for one model, but it can vary for different pre-trained models, as the number of available layers in each model is different. A certain number of individuals are randomly selected in the initialization process and get updated in the succeeding generations. In this paper, we set it to 10 individuals empirically. For each particular individual, two sets of features for both training and validation data are generated. These two feature sets contain the features from the same layer. Then, a Linear SVM classifier is built using the training data and its performance is further evaluated using the validation data. The evaluation process is called the fitness calculation, which generates a fitness score for each individual. The fitness score is calculated using the fitness function $f(i)$ (line 5) in Eq. 1, defined by the average F1 score (Avg. F1).

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

where C is the total number of classes in the target dataset, i presents each unique individual, and P_c and R_c are the precision and recall of class c, respectively. Precision represents the classifier's ability to not misclassify a positive sample, 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. Compared to using accuracy as the evaluation criterion, F1 score is more suitable for evaluating any dataset (balanced or imbalanced). A fitness score calculation needs to be done for every individual in each generation. Though extracting features from a pre-trained DL model is not the bottleneck of the process, the feature dimension is considered to affect the time complexity. To evaluate the overall feature strength of a layer in a certain DL model, each fitness score is calculated by fitting all features into a Linear SVM model. If a general machine learning model is built here to evaluate the features, it could bring the model complexity to be quadratic. However, utilizing an efficient implementation of linear SVM in scikit-learn (Pedregosa et al. 2011) can have the function scale almost linearly to millions of samples and/or features. Besides, the prediction complexity is $O(d)$, where d is the number of input dimensions since it is just a single inner product. Therefore, evaluating feature strength for every individual here will not result in higher computational complexity. Also, the fitness of an individual is evaluated based on the original feature dimension that was extracted from the DL model. In the layer selection and feature selection stage, the feature dimension stays unchanged.

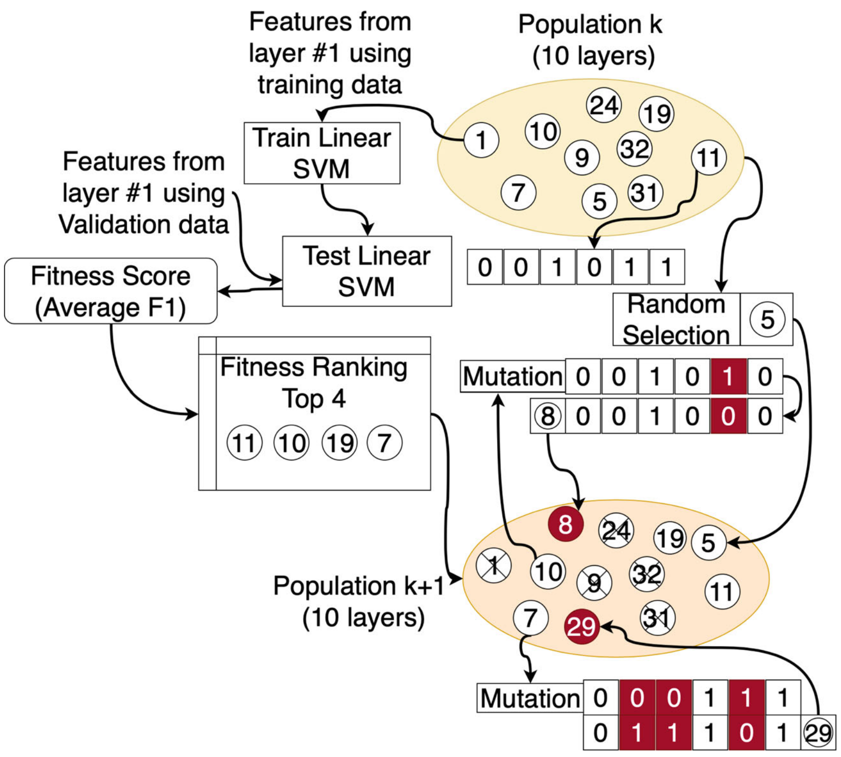

Fig. 2 Genetic code evolution example for one generation in the layer selection phase

The individuals in a specific population are ranked in a descending order to create a ranking list. Based on the predefined retaining rate, the individuals on the top of the list continue to contribute to the next generation and produce offsprings. Other individuals also have a small chance to survive, depending on the random selection process in lines 13 to 17. The random selection process defined in the algorithm is straightforward. The purpose of it is to give more opportunities to lower-ranking individuals to have a second chance to survive and mutate to an individual with superior performance. As the random selection happens before the mutation operation, the individuals selected which are ranked lower than other individuals also get a chance to flip several digits to get the gene representation closer to the current one, but carry a higher-ranking feature performance. Ignoring those individuals may end up denying some of the potential paths that lead to better solutions. All other individuals are removed from the population when preparing for the next generation.

Figures 2 and 3 explain the genetic code evolution process for one generation in the layer selection and feature selection phases, respectively. The figures depict two genetic operations as described through lines 13 to 24 in the algorithm.

3.2 Layer Selection and Feature Selection

In the layer selection phase, flipping one binary number (0 to 1 or 1 to 0) in the evolution process results in extracting a different level of features from another layer and consequently affects the performance of the classifier. For instance, the code changing from 001010 to 001000 means that, instead of layer #10, layer #8 is selected for feature extraction. This operation repeatedly applies to one selected individual in the mutation process. We allow this process to affect every position of the genetic code each time with a 0.5 mutation rate, which means it holds an equal chance to keep or flip every number.

The crossover operation is excluded while using the EP algorithm. When using the GA algorithm, this operation generates new individuals for the next population by combining the genetic codes from two retained individuals. By taking the mutation operation several times on one individual, we expect that the operation is good enough to be a replacement of the crossover operation for generating new individuals. Hence, multiple positions can be affected when the mutation is completed for producing a new individual.

Layer selection evolves the individuals for several generations to identify the top layer candidates. The top

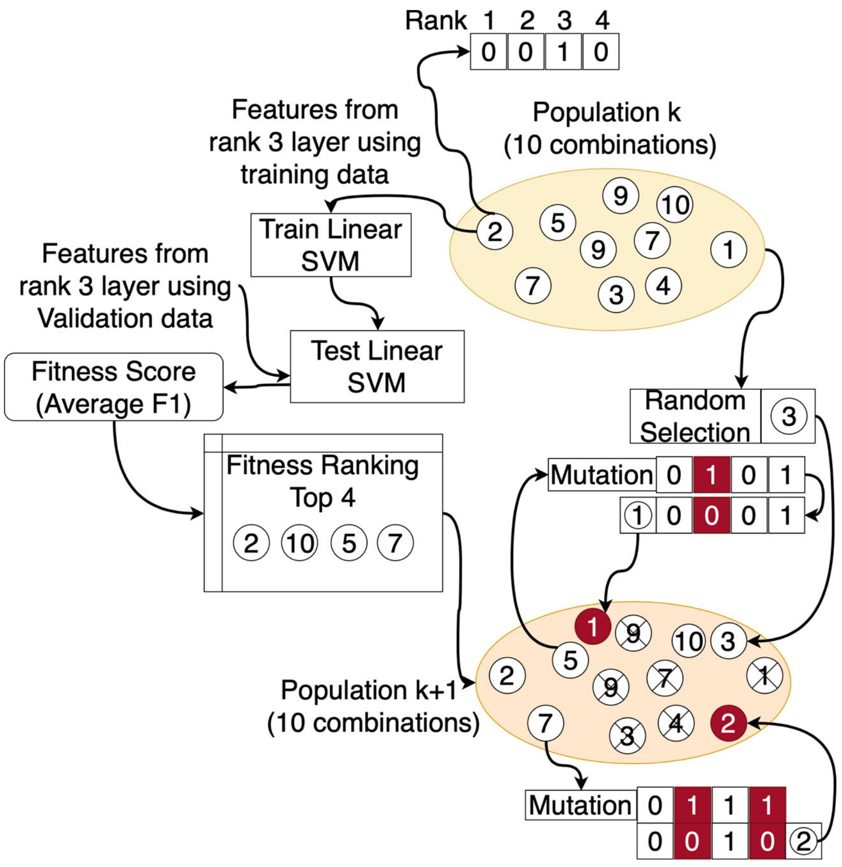
Rank 1 2 3 4
0 0 1 0
Features from rank 3 layer using training data
Population k (10 combinations)
Train Linear SVM
Features from rank 3 layer using Validation data
Test Linear SVM
Fitness Score (Average F1)
Fitness Ranking Top 4
2 10 5 7
Random Selection
Mutation 0 1 0 1
1 0 0 0 1
Population k+1 (10 combinations)
Mutation 0 1 1 1
0 0 1 0 2

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

portion of the individuals in the last generation presents a specific dataset with the most stable features. Those features are obtained by taking the predefined portion of the individuals from the final ranking list using the retaining rate. The feature sets are further encoded as different feature combinations to move forward with feature selection.

As shown in Fig. 3, the encoding strategy changes from taking a single feature layer to having a combination of the feature layer candidates as a specific genetic code. All genetic operations remain the same except that each digit means including or abandoning a candidate layer while forming the final feature set (e.g., a "0" means "exclude", while a "1" means "include"). Therefore, a mutation operation on one single position will either add or remove a feature layer from the final feature set.

For every pre-trained deep learning model, the output of the current phase is a combination of top feature sets. The selected combination produces the highest fitness score when running on a Linear SVM classifier using the same validation data. Only one feature model that holds the highest score among all candidate models will be selected as the best feature extractor which is used to build the final classification network in the next step. Though the total number of features is scaled up to a higher dimension when more layers are included in the potential feature set, validation of the performance of a feature set combination still can be done in linear time by using Linear SVM. Therefore, the complexity of the feature selection across multiple pre-trained models is still maintained in $O(N_{features} * d + N_{ind} log N_{ind})$, which is dominated by the fitness score ranking process regarding N_{ind} individuals if scaled to a large number of individuals or by the feature dimensions $N_{features}$.

3.3 Network Evolution

In the early sections, Algorithm 1 only selects the features that have the potential alternatively to represent new concepts that are initially unknown in the general models. Without TL, new concepts can not be captured. Therefore, constructing a specific model transfers the knowledge learned from the general model to the knowledge that conveys a new feature relationship with the target domain concepts. The specific models scale the features to lower dimensions while transferring the known information to an isomorphic that links a sequence of general concepts to target concepts. As the first part of the framework is selecting general features with partitions, the second part focuses on specific feature learning that provides feedback on the general model to be suitable for specific targets.

As shown in Algorithm 2, taking a similar process as Algorithm 1, network selection also starts with an initial population that is generated by random search. The network evolution process considers each network structure as a unique individual and leverages all the genetic operations as defined in the genetic code evolution process. In each generation, the genetic operations are triggered by assessing the similarities between the populations in subsequent generations. Specifically, distance calculation ensures that the individuals continue to improve while the top networks in the same generation are very similar. In each generation, top networks are selected based on a retaining rate. Those networks are responsible for producing offsprings that depict the new network structures. The same fitness function, as described in Eq. 1, is also used here to rank the networks (lines 2 to 5).

Algorithm 2 Network evolution.

```
1  $RETAIN \leftarrow 0.4, SELECT \leftarrow 0.1, MUTATE \leftarrow 0.5$
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     $OR\ \forall H_{xs} > Significant\ WHERE\ s \in parents$
       then
17         $parents.append(grade[x])$
18 # Mutation
19 $size \leftarrow Population.size - parents.size$
20 while $children.size < size$ do
21     select $parent$ randomly from $parents$
22     $child.network = parent.network$
23     for $hyperparameter\ in\ child.network$ do
24         # Mutate every hyperparameter
25         if $MUTATE > random()$ then
26             $MUTATE(child.network[hyperparameter])$
27     $children.append(child)$
28 $parents.extend(children)$
29 return $parents$
```

As the networks are initiated with randomly generated weights, the same network structure can perform each time differently. To only keep track of the best weights for each network, additional storage is utilized to record the performance of each network that has already been discovered. In addition, because different weights are applied to the same network structure, the same gene

Table 1 The available choices for network hyperparameters and the corresponding binary encoding digits

Hyperparameters	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

can appears multiple times in the ranking list. The more times a particular structure shows in the list, the higher the chance that the next generation's offsprings share substantial similarity between each network and this particular structure. Consequently, the evolving process might slow down and makes the solution held within a local optimal. Distance calculation (lines 6 to 10) is utilized to overcome this limitation, which plays a vital role in ensuring that the population can keep searching for more combinations in the later generations after identifying several network configurations with proper performances.

To calculate the distance between every two network structures in the current generation, the genetic encoding function (line 7) converts one set of choices of four candidate hyperparameters into a unique binary string. Table 1 lists the available choices of each hyperparameter and the corresponding length of the encoding digits. The Hamming distance calculation (line 9) is used to generate the distance matrices $H = (\sigma(x, y))$, where x and y are bounded by the total number of individuals P in one generation. The time complexity of calculating the Hamming distance between two individuals is at most linear, which is bounded by $O(n)$, where n is the total number of encoding digits. Therefore, a Hamming distance matrix for all individuals can be generated within $O(N_{ind} * n)$. Furthermore, an individual is defined as "significant" when the distance between itself and any higher-ranking individuals is greater than the current generation's mean distance. Therefore, the space of heuristics in each generation is refined by itself when the current generation covers either a narrow or broad searching space. Lines 11 to 27 illustrate the procedure of all the genetic operations (retaining selection, random selection, and mutation) within specified activating conditions (defined at line 1).

Instead of changing one bit of the genetic code, the mutation operator randomly chooses a value for one specific hyperparameter. Therefore, the number of position shifts in one mutation operation is not restricted to one. Since we have four hyperparameters, the mutation operation repeats at most four times, depending on the mutation rate. The probability of keeping one hyperparameter the same relies on the mutation rate of p. Therefore, consider making a one-time decision on one hyperparameter vs. making decisions on the binary representation, keeping one hyperparameter as a whole change of the mutation probability P. If a hyperparameter is encoded with e digits, there is a $(1 - p)$ probability of keeping the hyperparameter unchanged. However, this probability will be reduced to $(1 - p)^e$ if each digit is independent. Though each individual has a higher chance to stay unchanged after mutation, this change allows the heuristics to cover a sparse space and then mutate several bits during each decision. If considering each digit independently, the probability of changing more digits is always smaller than changing fewer digits. For instance, letting "101" mutate to "110" is more complicated than having "101" mutate to "111", as it causes two-bit mutations. This change could help the later evolution process when most of the individuals in the generations are very similar. Section 4.2 further details how this strategy meets our expectations.

4 Experimental Analysis

4.1 Experimental Setup

The proposed framework includes four pre-trained deep learning models as the feature model candidates. The model

Table 2 The pre-trained deep learning model candidates with the available number of feature choices

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

Table 3 The statistical information of the disaster dataset

No.	Concepts	Harvey	Irma
1	Demonstration	42	8
2	Emergency Response	81	20
3	Flood and Storm	426	177
4	Human Relief	70	1
5	Damage	42	172
6	Victim	75	16
7	Speak	347	63
Total		1083	457

name and the corresponding number of available layers for feature extraction are listed in Table 2. In the layer selection phase, the number of individuals in each population is empirically set to 10 with a 0.4 retaining rate r. Therefore, K^{10*r} choices of unique feature sets are available for each model, where K is the available number of layers for feature extraction. 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 search space is far too large to be explored exhaustively by hand.

Four datasets from different domains are selected to evaluate the proposed framework: two imbalanced and two balanced datasets. Tables 3 and 4 depict the statistical information of the imbalanced datasets. The disaster dataset (as shown in Table 3) contains Youtube videos collected during two major hurricane events in the year 2017: Harvey in Texas and Irma in Florida. In this dataset, the "Flood and Storm" concept contains most of the samples. By following a chronological order, the first event is used as the training and validation data, while the second event is the testing data. One keyframe image is extracted to represent each video. The Network Camera 10K dataset (as shown in Table 4) is a surveillance camera dataset that contains images captured from a variety of places. 20 percent of the data was separated into testing data. Moreover, 20 percent of the training data were again randomly selected to form the validation data for the fitness score calculation. Concept "Highway" is the majority class in this dataset.

CIFAR10 and MNIST-Fashion are the two balanced datasets included in the experiments. Both are well-known public datasets. CIFAR10 classifies objects and animals, and MNIST-Fashion can be considered as a replacement of the original MNIST dataset for benchmarking machine learning algorithms. These two datasets already come with separated training and testing sets, but 20% of random training samples are still used to serve as the validation data for fitness score calculation during the evolution process. Both datasets include an equal number of samples for each class. CIFAR10 has concepts related to several objects (e.g., airplane, automobile, ship, and truck) and animals (e.g., bird, cat, deer, dog, frog, and horse), while 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.

4.2 Experimental Results

The performance of our proposed framework is compared with those of the three feature selection algorithms mentioned in the related work. Each of those algorithms selects a pre-trained model as the best feature model. BO is included here to assess its advantage regarding probability assumptions on our specific task. GA is included in the

Table 4 The statistical information of the Network Camera (NWC) 10K dataset

No.	Concepts	Instances	No.	Concepts	Instances
1	Intersection	855	8	Yard	161
2	Sky	495	9	Forest	139
3	Water Front	978	10	Street	431
4	Building+Street	603	11	Parking	99
5	Park	499	12	Building	243
6	Mountain View	719	13	Highway	3724
7	City	432	14	Park+Building	149
Total			9527		

Table 5 Proposed framework's final model performance on four datasets comparing to Bayesian optimization, genetic algorithm, and evolutionary programming based feature selection methods

Datasets	Methods	Model	Precision	Recall	W. Avg. F1
Disaster	BO-based feature	InceptionV3	0.3215	0.3256	0.3920
	GA-based feature	ResNet50	0.3212	0.3276	0.4430
	EP-based feature	ResNet50	0.3186	0.3440	0.4432
	Proposed Method	ResNet50	0.4496	0.3869	0.5548
Network Camera 10K	BO-based feature	ResNet50	0.6398	0.6263	0.7827
	GA-based feature	InceptionV3	0.6508	0.6339	0.7985
	EP-based feature	InceptionV3	0.6582	0.6405	0.7957
	Proposed Method	InceptionV3	0.6818	0.6183	0.8019
CIFAR10	BO-based feature	ResNet50	0.8949	0.8943	0.8945
	GA-based feature	ResNet50	0.9063	0.9061	0.9061
	EP-based feature	ResNet50	0.9073	0.9069	0.9070
	Proposed Method	ResNet50	0.9192	0.9192	0.9190
MNIST-Fashion	BO-based feature	ResNet50	0.9260	0.9263	0.9260
	GA-based feature	ResNet50	0.9289	0.9292	0.9289
	EP-based feature	ResNet50	0.9294	0.9298	0.9294
	Proposed Method	ResNet50	0.9340	0.9340	0.9339

comparison to determine whether the mutation operation in the EP method is powerful enough to replace the crossover operation when converging to an optimal solution.

In Table 5, the proposed method is compared with the other three feature selection methods on the four datasets. Three metrics were considered, namely Precision, Recall, and Weighted average F1 score (W. Avg. F1). As the F1 score captures the trade-off between precision and recall, it is more suitable to evaluate the overall model performance. Also, the F1 score is a better measurement when evaluating the performance of an imbalanced dataset comparing to accuracy. In the comparison, the three feature selection methods are only used to select the final feature sets and evaluated with the Linear SVM models. Only our proposed method reports the final model performance incorporating the EP-based feature selection method and then training on a lately created neural network model. As shown in Table 5, the GA-based and EP-based feature selection methods always pick the features from the same model for each dataset. Thus, we proved that the mutation operation is good enough to replace the crossover operation in the evolution process. For the disaster dataset, our proposed framework selected the ResNet50 model to extract the features and further enhanced the model performance by constructing a better network model that can capture more information from the features. This enhancement increases the feature performance by more than 10% when compared to the others using the W. Avg. F1 value. For all the other datasets, the final models' performances are also improved. For the two balanced public datasets, CIFAR 10 and MNIST-Fashion, though all the methods selected the same pre-trained model, the features strength are not all the same. Our proposed framework can bring CIFAR 10 and MNIST-Fashion data performances beyond 90% and further enhance both of them to reach a better performance by building new feature learning models.

Figure 4 illustrates the single model's performance using three feature selection methods, where EP-based feature selection (last bar in the figure for each comparison) is leveraged in the proposed framework. The y-axis compares the scores of the evaluation metrics (ranging between 0 and 1). Though MobileNet model is not selected by any of the methods for the four experimental datasets, this

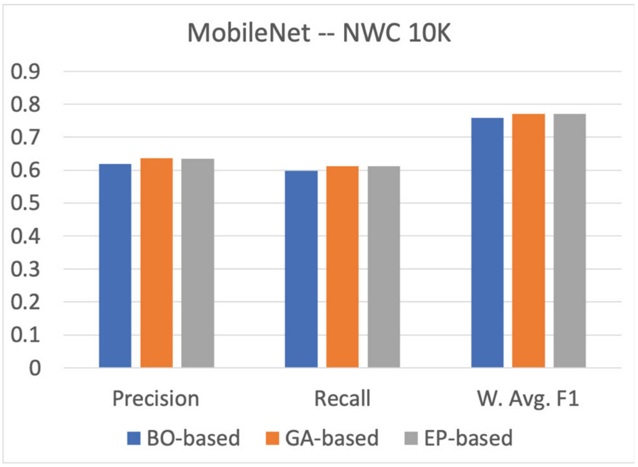

Fig. 4 MobileNet model feature performance on NWC 10K dataset

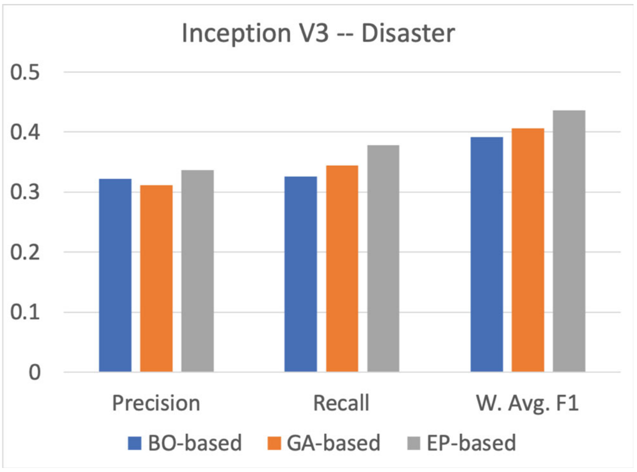

Fig. 5 Inception V3 model feature performance on Disaster dataset

comparison here is to show that our proposed framework could identify the best feature set in the feature selection phase by including any specific model candidates. Figure 5 also shows that the proposed framework identifies the best feature set from the InceptionV3 model. However, it is not selected as the best model. Thus, it is true that this model's best feature set's performance cannot compete with the feature set from the ResNet50 model as shown in the experimental result table. From these bar charts, it can be seen that the proposed framework always obtains the feature set that performs the best considering all the evaluation metrics.

In addition to evaluate the final model performance, the efficiency of the proposed framework during network structure evolution is also observed. Figure 6 depicts the evolutionary process by comparing the averaged fitness scores (average F1 scores of the validation data) of the top retained genes in each generation with the top 1 individual. Defined by the retaining rate, only the top 40% of individuals can survive and continue to contribute to

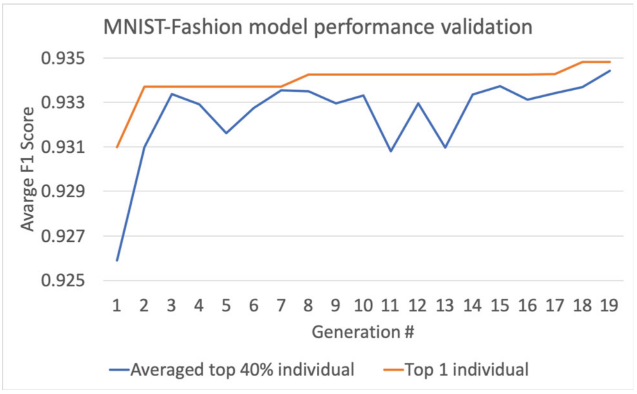

Fig. 6 The comparison of top individual's average F1 score in each generation

the next generation. As can be seen from the plot, the performance of the top 1 individual in each generation has steadily increased and reached a certain optimal F1 score near the 8th generation. After that, the new populations in each generation keep searching for a better solution and successfully exceed the optimal score in the 18th generation.

4.3 Complexity Analysis

The time complexity analysis of the proposed framework can be separated into three parts: (1) layer selection, (2) feature selection, and (3) final model construction. Each part has its own set of genetic operations which requires running time to be specifically associated with the size of the population N_{ind}, the length of the gene L, and the feature dimension d. The complexities of the first two parts are similar, as both of them only utilize the pre-trained models with fixed feature weights to extract general features.

4.3.1 Layer Selection and Feature Selection

Ideally, the genetic code evolution process can have all individuals (feature model) run in parallel. Therefore, the running time for operations involving only a single individual can always be scaled up to finish the operations for the entire population. The bottleneck is only restricted by the operation that needs all individuals in the calculation.

- **Fitness calculation**: Normally, the time complexity of the entire evolution process mainly depends on the fitness function. In the proposed framework, incorporating a Linear SVM model enables this calculation to be scaled up nearly linearly to a large number of features or training instances. Therefore, the fitness score calculation for the entire population can be done in $O(N_{features} * N_{instances})$ with a cluster to process each individual in parallel.
- **Selection operation**: Selection operation needs all individuals included in one calculation, which sorts the individuals in descending order and makes retaining decisions on each individual. This bottleneck operation can be done in $O(N_{ind} log N_{ind})$.
- **Mutation operation**: Mutation operation is applied to every bit of each specific gene in the first two parts. This operation can also be done in parallel as any two individuals do not affect each other. The mutation operation for each generation will be activated at most $O(L)$ times.
- **Crossover operation (GA only)**: This operation is included in GA-based feature selection. In our proposed framework, we reduced the dependency of individuals within each generation as only the mutation operation is included. If the crossover operation is included, additional time needs to be consumed for offspring

generation. This operation cannot be run in parallel as it needs to select two individuals each time from the retained population. Time complexity for this particular operation is $O(N_{ind})$.

4.3.2 Network construction

Building new network models using the features selected from the previous steps consumes most of the running time in the end. In the proposed framework, we consider building smaller neural network models as an extension of the general feature model, which only contain a few numbers of dense layers. Dense layers, also known as fully connected layers, can dominate the parameters of the entire network models with a computation complexity of $O(N^2)$, where N is the max number of neurons in one layer.

- **Fitness calculation**: In the network construction part, the networks selected in each generation are evaluated with a small number of training epochs with early stop. Thus, the fitness calculation complexity for each generation is bounded by $O(N_{max}^2)$. Each network is independent and can be run in parallel. As mentioned in Table 1, the N_{max} is 1024 for our candidate models.
- **Selection operation**: Selection operation still depends on sorting all fitness scores which can be performed with a complexity of $O(N_{ind} log N_{ind})$.
- **Mutation operation**: Mutation operation applies to every network structure hyperparameter instead of every binary gene code of each individual. This operation can be done in parallel for each individual while the complexity is reduced from $O(L)$ in the first two parts to $O(N_{hyper})$ here. The total number of hyperparameters that can be changed in an individual is always not higher than the length of the gene.

5 Conclusions

In this paper, the possible challenges of applying pre-trained deep learning models on different target problem domains are first identified. Then, a generalized framework using evolutionary programming is proposed to automatically determine the best feature set from a group of model candidates and construct a new network model that takes those selected features as the input to better understand the correlations between the features and the target concepts. Hence, the most representative features for a specific target domain are further transformed using a new classifier to enhance the final model's performance. The experimental results have shown that our proposed framework outperforms the other algorithms while identifying the best feature set regardless of the model candidate change. Overall, it is demonstrated that a time-consuming task, such as deep network construction conducted by experts, can be done by an automated process that surpasses the human ability and reaches an optimal solution effectively.

Acknowledgements This research is partially supported by NSF CNS-1952089 and OIA-1937019.

References

Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford: Oxford University Press.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(Feb), 281–305.

Chang, W.L. (2019). The impact of emotion: a blended model to estimate influence on social media. *Information Systems Frontiers*, *21*(5), 1137–1151.

Chen, C., Zhu, Q., Lin, L., Shyu, M.L. (2013). Web media semantic concept retrieval via tag removal and model fusion. *ACM Transactions on Intelligent Systems and Technology*, *4*(4), 61.

Chen, S.C., & Kashyap, R.L. (2001). A spatio-temporal semantic model for multimedia database systems and multimedia information systems. *IEEE Transactions on Knowledge and Data Engineering*, *13*(4), 607–622.

Chen, S.C., Rubin, S.H., Shyu, M.L., Zhang, C. (2006). A dynamic user concept pattern learning framework for content-based image retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *36*(6), 772–783.

Chen, S.C., Shyu, M.L., Kashyap, R. (1998). Augmented transition network as a semantic model for video data. *International Journal of Networking and Information Systems*, *3*(3), 9–25.

Chen, S.C., Shyu, M.L., Zhang, C. (2005). Innovative shot boundary detection for video indexing. In *Video data management and information retrieval* (pp. 217–236). IGI Global.

Chen, S.C., Shyu, M.L., Zhang, C., Kashyap, R.L. (2001). Identifying overlapped objects for video indexing and modeling in multimedia database systems. *International Journal on Artificial Intelligence Tools*, *10*(04), 715–734.

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In *The IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications. CoRR arXiv:1704.04861.

Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q. (2017). Densely connected convolutional networks. In *The IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).

Lameski, P., Zdravevski, E., Mingov, R., Kulakov, A. (2015). 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* (pp. 464–474): Springer.

Li, H., He, F., Liang, Y., Quan, Q. (2019). A dividing-based many-objective evolutionary algorithm for large-scale feature selection. *Soft Computing*, 1–20.

Li, X., Chen, S.C., Shyu, M.L., Furht, B. (2002). Image retrieval by color, texture, and spatial information. In *The 8th international conference on distributed multimedia systems* (pp. 152–159).

Lin, L., & Shyu, M.L. (2010). Weighted association rule mining for video semantic detection. *International Journal of Multimedia Data Engineering and Management (IJMDEM)*, *1*(1), 37–54.

Mania, H., Guy, A., Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. CoRR arXiv:1803.07055.

Mitchell, T.M. (1997). Machine learning.

Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J. (2016). Pruning convolutional neural networks for resource efficient transfer learning. CoRR arXiv:1611.06440.

Mukherjee, S. (2020). Emerging frontiers in smart environment and healthcare–A vision. *Information Systems Frontiers*, *22*(1), 23–27.

Pan, S.J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011). Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M.P., Shyu, M.L., Chen, S.C., Iyengar, S. (2018). A survey on deep learning: algorithms, techniques, and applications. *ACM Computing Surveys*, *51*(5), 92.

Pouyanfar, S., Tao, Y., Tian, H., Chen, S.C., Shyu, M.L. (2019). Multimodal deep learning based on multiple correspondence analysis for disaster management. *World Wide Web*, *22*(5), 1893–1911.

Pouyanfar, S., Yang, Y., Chen, S.C., Shyu, M.L., Iyengar, S. (2018). Multimedia big data analytics: a survey. *ACM Computing Surveys*, *51*(1), 10.

Shin, H.C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., Summers, R.M. (2016). Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, *35*(5), 1285–1298.

Snoek, J., Larochelle, H., Adams, R.P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951–2959).

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *The IEEE conference on computer vision and pattern recognition* (pp. 2818–2826).

Tian, H., & Chen, S.C. (2017). Mca-nn: multiple correspondence analysis based neural network for disaster information detection. In *IEEE third international conference on multimedia big data (bigMM)* (pp. 268–275): IEEE.

Tian, H., Chen, S.C., Shyu, M.L. (2019). Genetic algorithm based deep learning model selection for visual data classification. In *The 20th international conference on information reuse and integration for data science* (pp. 127–134): IEEE.

Tian, H., Chen, S.C., Shyu, M.L., Rubin, S. (2019). Automated neural network construction with similarity sensitive evolutionary algorithms. In *The 20th international conference on information reuse and integration for data science* (pp. 283–290): IEEE.

Tian, H., Pouyanfar, S., Chen, J., Chen, S.C., Iyengar, S.S. (2018). Automatic convolutional neural network selection for image classification using genetic algorithms. In *The IEEE international conference on information reuse and integration* (pp. 444–451): IEEE.

Tian, H., Tao, Y., Pouyanfar, S., Chen, S.C., Shyu, M.L. (2019). Multimodal deep representation learning for video classification. *World Wide Web*, *22*(3), 1325–1341.

Tian, H., Zheng, H.C., Chen, S.C. (2018). Sequential deep learning for disaster-related video classification. In *IEEE conference on multimedia information processing and retrieval (MIPR)* (pp. 106–111): IEEE.

Wang, D., & Zheng, T.F. (2015). Transfer learning for speech and language processing. In *The asia-pacific signal and information processing association annual summit and conference* (pp. 1225–1237): IEEE.

Winston, P.H. (1992). Artificial intelligence.

Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection* (pp. 117–136): Springer.

Yao, X., Liu, Y., Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, *3*(2), 82–102.

Zhu, Q., Lin, L., Shyu, M.L., Chen, S.C. (2011). Effective supervised discretization for classification based on correlation maximization. In *IEEE international conference on information reuse & integration* (pp. 390–395): IEEE.

Zhu, W., Cui, P., Wang, Z., Hua, G. (2015). Multimedia big data computing. *IEEE Multimedia*, *22*(3), 96–c3.

Haiman Tian received the B.S. degree in computer science from Sun Yat-Sen University, Guangzhou, China, in 2009, the M.S. degree in computer engineering, and the Ph.D. degree in computer science from Florida International University, Miami, FL, USA, in 2014 and 2019, respectively. Her research interests include multimedia data mining, machine learning, deep learning, multimodal data analytics, and disaster information management.

Shu-Ching Chen is a Professor and Associate Director in the School of Computing and Information Sciences (SCIS), Florida International University (FIU), Miami. He received his Ph.D. degree in Electrical and Computer Engineering and Master's degrees in Computer Science, Electrical Engineering, and Civil Engineering, all from Purdue University, West Lafayette, IN, USA. He has authored and coauthored more than 360 research papers in journals, refereed conference/symposium/workshop proceedings, book chapters, and three books. Dr. Chen was named a recipient of the ACM Distinguished Scientist Award. He received the best paper awards from the IEEE International Conference on Information Reuse and Integration and the IEEE International Symposium on Multimedia. He is a fellow of IEEE, AAAS, and SIRI.

Mei-Ling Shyu is a Professor and Associate Chair at the Department of Electrical and Computer Engineering (ECE), University of Miami (UM). She received her Ph.D. degree from the School of Electrical and Computer Engineering and Master degrees in Computer Science, Electrical Engineering, and Restaurant, Hotel, Institutional, and Tourism Management, all from Purdue University, West Lafayette, IN, USA. She has authored and co-authored 2 books and more than 290 research papers in journals, refereed conference/symposium/workshop proceedings, book chapters. Dr. Shyu was awarded the Computer Society Technical Achievement Award and the ACM Distinguished Scientists Award. She received four Best Paper Awards and one Best Student Paper Award with her student, all from IEEE conferences. She is a Fellow of IEEE, AAAS, and SIRI.