2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

Evolutionary Generative Contribution Mappings

Masayuki Kobayashi
Yokohama National University
Kanagawa, Japan
kobayashi-masayuki-xc@ynu.jp

Satoshi Arai
Yokohama National University
Kanagawa, Japan
arai-satoshi-rw@outlook.jp

Tomoharu Nagao
Yokohama National University
Kanagawa, Japan
nagao@ynu.ac.jp

***Abstract*—Although convolutional neural networks (CNNs) have significantly evolved and demonstrated outstanding performance, their uninterpretable nature is still considered to be a major problem. In this study, we take a closer look at CNN interpretability and propose a new method called Evolutionary Generative Contribution Mappings (EGCM). In EGCM, CNN models incorporate both a classification mechanism and an interpreting mechanism in an end-to-end training process. Specifically, the network generates the class contribution maps, which indicate the discriminative regions for the model to identify a specific class. Additionally, these maps can be directly used for classification tasks; all that is needed is a global average pooling and a softmax function. The network is represented by a directed acyclic graph and optimized using a genetic algorithm. Architecture search enables EGCM to deliver reasonable classification performance while maintaining high interpretability. We apply the EGCM framework on several datasets and empirically demonstrate that the EGCM not only achieves excellent classification performance but also maintains high interpretability.**

***Index Terms*—convolutional neural network, visualization, designing neural network architectures, deep learning**

I. Introduction

Convolutional neural networks (CNNs) have been successfully employed in many computer vision tasks [1], [2], and evolved significantly. Despite their enormous success, however, the complexity of deep networks makes the interpretation of their models difficult and thus they are frequently used as black-box predictors [3], [4]. Their uninterpretable nature may diminish users' trust and can be a barrier to their adoption in applications. For instance, in applications where interpretability is important, including medical decision making, users do not want to rely on black-box systems.

Traditionally, it is assumed that there is a trade off between learning capacity and interpretability [5], [6]. Especially, the CNN models can achieve the state-of-the-art performance, but we have to compromise on their interpretability. As CNN models have become deeper and more complex, they have become more difficult to interpret [7]. Therefore, it is important to review work on the trade off between CNN interpretability and its performance.

A natural question arise from this: *how can CNN models maintain high interpretability without sacrificing their performance?* In this paper, we tackle this problem from the perspective of the network architecture search. To this end, we propose *Evolutionary Generative Contribution Mappings* (EGCM) for achieving a high classification performance together with high interpretability. In EGCM, the networks incorporate both a classification and an interpreting mechanism in an end-to-end training manner. Specifically, the network generates class contribution maps that directly indicate the discriminative regions that allow the model to identify a specific class; see Figure 1. The network is represented by a directed acyclic graph and optimized by a genetic algorithm (GA) [8], [9]. We applied the EGCM framework to several datasets to investigate its learning capacity and class discriminativeness. We empirically show that EGCM maintains high interpretability without sacrificing classification performance.

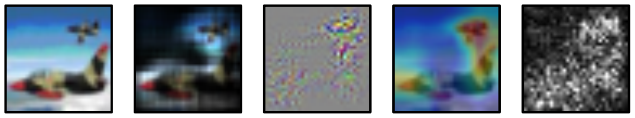


Fig. 1. (a) Original image. (b) Class contribution maps (ours). (c-d) Various visualizations for our CNN (architecture in supplementary material).

II. Related Work

We briefly review the two directions of most related work: visualizing CNN predictions and automatic network architecture design.

A. Visualizing Convolutional Neural Network Predictions

In a number of previous studies, CNN predictions were visualized to improve their interpretability. Most of these methods visualize the CNN's predictions by highlighting which input pixels are more responsible for the classification decisions [7], [10]–[12], [12]–[18]. These methods are efficient for interpreting *pretrained CNN* predictions; however, interpretability (or quality) of visualizations depend greatly on the choice of techniques (i.e. different approaches yields different visualizations for the same networks). In addition, as Adebayo *et al.* [19] reported, some existing saliency methods were independent of both the model parameters and the data that the model was trained on. This implies that the visualization generated by these studies are not always related to the classification decisions. In our method, however, we attempt to build CNN models whose visualizations are used directly for classification tasks

In parallel, in several studies network architectures for understanding and interpreting CNN classification decisions were designed [20]–[25]. These methods are efficient for interpreting a CNN's classification decisions. However, as Selvaraju *et al.* [7] mentioned, these methods may trade off the model performances for their interpretability. In addition, their methods may lack the ability to provide the fine-grained (i.e. pixel-level) importance score for classifications. For instance, Zhou *et al.* [20] proposed a technique called class activation mapping (CAM) for visualizing the class discriminative regions. Although CAM maps are very useful for localizing the important regions, their resolution is low, which makes it difficult to interpret CNN classification decisions. In particular, considering the classification of smaller images, the resolution of visualizations is more crucial. Our visualizations provide the pixel-level importance scores to capture the details in the image. In addition, we employ the network architecture search technique to find an appropriate network architecture for our mechanism. This helps to deliver a high classification performance together with high interpretability.

In this paper, however, we revisit these methods for the following reasons. First, the interpretable visualizations should be directly related to the classification decisions. Most prevailing visualization techniques are efficient in terms of understanding a CNN's operation; however, their visualizations are heuristic and sometimes they are not directly related to the classification decisions [19]. Finally, in our opinion the gap between learning capacity and interpretability has been narrowing. Recent techniques for automating the design of network architectures have achieved promising performances [26], [27]. By exploiting the potential of these techniques, we build a model that does not trade off the model's performance for its interpretability.

B. Automatic Network Architecture Design

The design of the network architecture is one of the most important tasks for improving the performance, and many automatic architecture design methods have been proposed.

One approach for automatic architecture design is to use evolutionary algorithms. Neuroevolution is one of the earliest approaches in this field. It attempts to optimize the network topologies together with their weights [28]–[30]. In recent studies, however, network architectures were designed by an evolutionary algorithm and their weights were trained by a stochastic gradient descent method through back-propagation [31], [32].

Another recent trend is to use reinforcement learning; Q-learning and policy gradient methods are employed to search for better network architectures [26], [33], [34]. Although these approaches continue to show significant advantages over human-designed models, they are computationally expensive; e.g. 450-800 GPUs were used for the architecture search presented in [26], [34]. Therefore, a new research direction for improving the efficiency of the network architecture search has appeared [27], [35], [36].

In this study, we employed the CGP encoding scheme presented in [32], [37]. The main advantages of this encoding scheme are its flexibility, extensibility, and computational efficiency: it can encode and optimize various types of network architecture using only a small amount of resources. In the GA approaches, only appropriate coding (*genotype*) is needed, while the design of the search space for network architecture searches needs to be considered in most prevailing reinforcement approaches.

III. Evolutionary Generative Contribution Mappings Method

Evolutionary Generative Contribution Mappings (EGCM) was inspired by the attention mechanism and recent advances in deep neural networks, including the automatic network architecture search. EGCM comprises two core techniques: (1) the generation of a *class contribution map* by means of identifying the class discriminative regions, and (2) the use of a GA to search for good network architectures.

A. Class Contribution Map Generation

In EGCM, the networks are fully end-to-end trainable and incorporate both a classification mechanism and an interpreting mechanism such that the network generates visualizations that can be used not only for classification, but also for interpreting CNN classification decisions. To achieve this, we consider EGCM to be a *large* attention module that calculates the response at every pixel in the input, where the weights are trained in an end-to-end manner.

In a *prevailing* feedforward convolutional network, where we denote the convolutional network by $\phi_{\mathrm{CNN}}(\cdot)$ and x_0 is a single input image. The output y is computed as

$$y = \phi_{\mathrm{CNN}}(x_0) \tag{1}$$

In contrast, EGCM first defines the weight maps W and computes the three-dimensional maps $M \in \mathbb{R}^{m\times n\times v}$ of width m, height n and channel v:

$$M^c = x_0 \odot W^c \quad s.t.\ W = \phi_{\mathrm{EGCM}}(x_0) \tag{2}$$

where c is the class index; $\odot$ denotes element-wise multiplication. We consider the network $\phi_{\mathrm{EGCM}}(\cdot)$ to be built on several convolutional and deconvolutional layers. We refer to W as *class weight maps*, which express the pixel-level importance score for classifications. The outputs M are three-dimensional maps called *class contribution maps* (CCM) that can be used not only as indicators of which pixels of the input are more discriminative, but also for classification.

Finally, a global average pooling is performed, followed by softmax classification to reach classification decisions:

$$y_c = \underbrace{\frac{1}{mnv}\sum_i\sum_j\sum_k M^c_{i,j,k}}_{\text{global average pooling}} \tag{3}$$

$$y = \mathrm{softmax}(y) \tag{4}$$

B. Architecture Search with Genetic Algorithm

In GA optimization, the genetic representation of the problem must be defined. In EGCM, we follow the method in [32], [37] and use the CGP encoding scheme for architecture representation. As in the standard CGP encoding scheme,

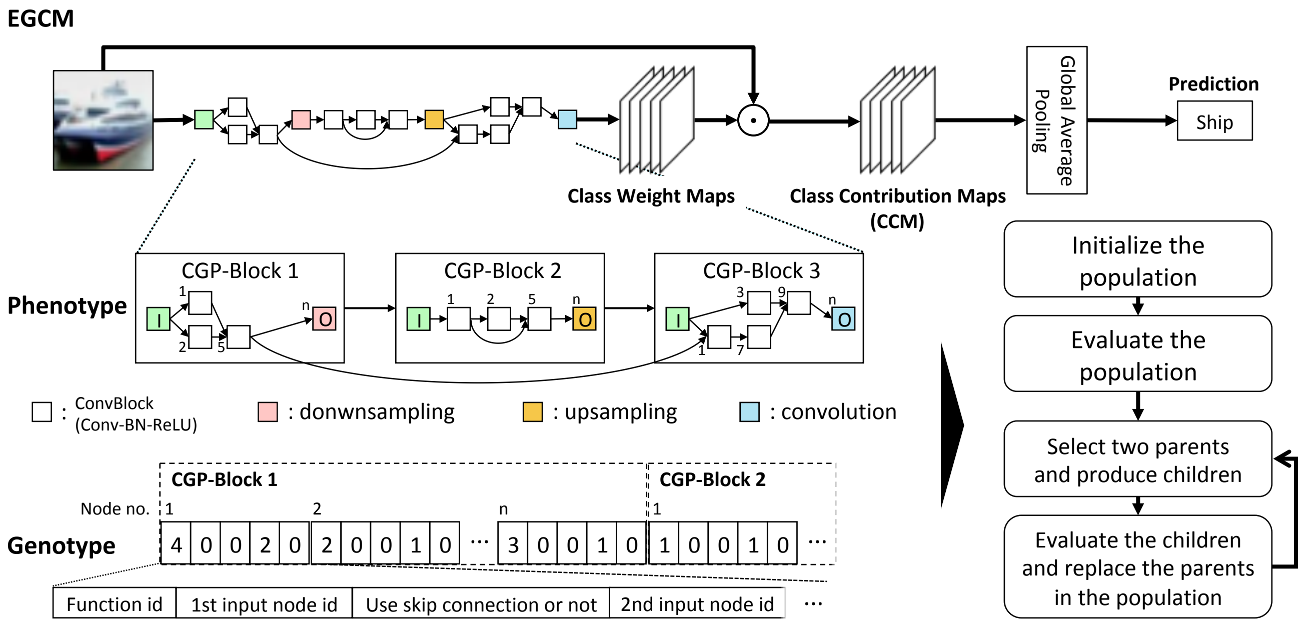


Fig. 2. *Evolutionary Generative Contribution Mapping* (EGCM) with a 3-layer CGP-Block. The network architecture is represented by multiple graphs (*phenotype*) and is encoded by a string of integers (*genotype*). An overview of our genetic algorithm is also illustrated.

we represent the network architecture by a directed acyclic graph with a two-dimensional grid. This graph (*phenotype*) is encoded by a string of integers (*genotype*) and optimized by a GA. Figure 2 illustrates the network representation, a genotype and the corresponding phenotype, which are described in detail later.

1) Network Representation: In EGCM, the class contribution maps are computed by element-wise multiplication of the class weight maps with the input images. Therefore, the class weight maps and the input images must be the same size; otherwise, Equation (2) is not viable. However, the most frequently used CNN architectures consist of several pooling layers to reduce the size of the representation. To keep the size of the output the same as that of the input image but also utilize downsampling and upsampling in our architecture, we confine the search space by dividing the network into several blocks, called *CGP-block*s; see Figure 2.

The CGP-blocks can be viewed as sub-graphs of a larger CGP graph. Each node in a CGP-block represents a certain type of function (the functions are described later) and is allowed to be connected only with nodes in the same block. This allows convolutional filters to be applied multiple times to extract complex and representative features at every spatial scale.

Furthermore, to incorporate the promising modules in our architecture, such as skip connections, shortcut connections, and branching layers, we introduce a hyper-parameter called the skip probability r_{skip}. This probability is the probability of allowing a node in the second half of the CGP-blocks to connect to a node in the first half of the CGP-blocks at the beginning of the GA. To be specific, each node may connect to a node in the opposite CGP-block, e.g. CGP-block 1 and 3 in Figure 2. The skip probability controls not only the connectivity of nodes but also the scheme for architecture search; that is, the use of large skipping probabilities allows the GA to start with smaller networks, while that of small skipping probabilities allows it to start with larger networks.

The output node of each CGP-block, which changes the size of the feature maps by performing downsampling or upsampling, is referred to as the *transition node*. To fix the number of channels, we use a convolution operation instead of the transition node at the last CGP-block.

2) Node Functions: In the CGP formulation, the types of node functions used in CGP are defined by the user. Referring to the modern CNN architectures, we select the following functions as node functions.

- **Basic function set:** The basic function set comprises the following functions: Convolution module (which are described in detail later), Concatenation, and Summation. Convolution module performs certain convolution operations. Each operation has an arbitrary number and size of filters, while the stride is fixed to be 1. We zero-pad the input feature maps so that the size of the outputs does not change. Concatenation is an operation that concatenates two feature maps in the channel dimension. Summation is an operation that performs element-wise summation of two feature maps. When summing two feature maps having a different number of channels, we expand the channels by padding the smaller feature map with zeros.
- **Downsampling function set:** The downsampling function set comprises the following functions: 2×2 max pooling, 2×2 average pooling, and Convolution module, in which the input feature maps are downsampled with a stride of 2.

- **Upsampling function set:** We use a fractionally strided convolution to upsample the input feature maps. Specifically, we introduce a function called DeconvBlock, which consists of a deconvolutional layer followed by batch normalization and the ReLU activation. DeconvBlock has an arbitrary number and size of filters while the stride is fixed to be 2.

3) Convolution Modules: We define two types of the Convolutional modules called ConvBlock and ShakeShakeBlock. ConvBlock consists of three operations: convolution followed by batch normalisation [38] and rectified linear units (ReLU) [39]. ShakeShakeBlock is one of the powerful residual modules introduced in [40] . We compare the classification performance by the use of the highly functional modules.

4) Phenotype: The phenotype is represented by a collection of directed acyclic graphs having N_r rows and N_c columns. Therefore, each CGP-block has $N_r \times N_c$ intermediate nodes and one transition node. The standard CGP has a parameter called level-back L, which controls the connectivity of the graph; the given nodes in the c-th column are allowed to be connected only from the $c - L$ to $c - 1$-th columns' nodes. This makes the graph directed and feedforward.

5) Genotype: The genotype encodes all CGP-blocks with a string of integers (*gene*), each of which specifies the attributes of a node. One gene is the function id, which determines the type of node function together with the number of channels F and the kernel size k. Another gene is the node id, representing which node is connected to this node as input. The remaining gene specifies whether or not the node uses skip connections and is initialized following the skip probability.

C. Genetic Algorithm (GA)

In general, a GA can be implemented in many ways. In this study, we used the minimal generation gap (MGG) model [41] as the generation alternation model. In this model, two individuals are randomly selected from the population and generate λ children at each generation. λ children are generated from two parents by applying certain genetic operations, such as mutation and crossover. Each child is trained using training dataset $\mathcal{T}$ and evaluated based on validation dataset $\mathcal{V}$. The validation accuracy is called *fitness*, and we search for a better network architecture to maximize this value.

At each generation, two individuals are selected from the parents and children—the best individual and that chosen by tournament selection—and replace the parents in the population. This method allows an efficient architecture search using only a small amount of resources but also allows population diversity to be maintained.

IV. Experimental Results

In this section, we describe several experiments that we conducted and demonstrate the efficiency of EGCM. We applied our method on CIFAR-10 and the Street View House Number (SVHN) dataset, and compared the classification performance with that of well-known CNN models. We also introduce a new dataset, called two-digit MNIST, and evaluate the class discriminativeness of our method.

A. Datasets

1) CIFAR-10: The CIFAR-10 dataset [42] consists of 60,000 color images in 10 classes. This dataset is split into a training set of 50,000 images and a test set of 10,000 images. We followed the method in [26], [33] and randomly selected 45,000 images from the training set for training the CNN; the remaining 5,000 images were used as the validation set.

2) Street View House Number: The SVHN dataset [43] consists of 32×32 color images of house numbers. It contains 73,257 training images, 26,032 test images, and 531,131 additional images, which can be used as an additional training set. To find a good architecture in reasonable time, we used only the original training set during the architecture search. Five thousand randomly sampled original training images were used as the validation set, and the remaining images were used for training the CNN.

3) Two-digit MNIST: The EGCM can be viewed as generating attention maps that identify the class discriminative region relevant to the classification and ignoring other regions. This signifies that the model can be trained to identify a category even if the image contains multiple objects. In order to investigate the class discriminativeness of EGCM, we generated a two-digit MNIST dataset by stacking two digits from the MNIST dataset [1]. In the two-digit MNIST, each image contains one two-digit number, but the second digit is assigned as a true label; i.e. this is a dataset for identifying the second digit in the image. For each digit in MNIST, we randomly sampled one digit with a fixed random seed and stacked them to generate 60,000 images for the training set and 10,000 images for the test set. The stacked images were resized to 28×28 by linear interpolation, and random rotation was performed. We followed the same approach as on SVHN and five thousand randomly sampled original training images were used as the validation set; the remaining images were used for training the CNN.

B. Training Settings

The training settings are similar to those provided in [49]. We initialized the weights by using the method in [50], and used the SGD with a Nesterov momentum [51] of 0.9, mini-batch size of 64, weight decay of 1.0×10^{-4}, and initial learning rate of 0.1. On CIFAR-10, SVHN, and two-digit MNIST, each network was trained for 50, 25, and 25 epochs, respectively, in which the learning rate was reduced by a factor of 10 at 50% and 75% of the total number of epochs. In addition, in order to reduce the time required for training the networks during the initialization, we used the smaller dataset, which consisted of 4,000 randomly sampled training images.

To determine clearly whether the classification performance is affected by pre-processing rather than by the use of EGCM, we divided the pixel values by 255 and used no pre-processing. We used only a standard data augmentation scheme, where 4 pixels are padded on each side and an input patch is randomly cropped from the padded image. On CIFAR-10, we additionally performed random horizontal flips. Following the method in [26], we trained each CNN for a specific number

TABLE I
RESULTS ON THE CIFAR-10 AND SVHN DATASETS. WE RAN OUR METHOD THREE TIMES AND REPORTED THE CLASSIFICATION ERRORS.

Model	**No. of Parameters** ($\times 10^6$)	**CIFAR-10**	**SVHN**
Maxout [44]	–	9.34	2.47
Network in Network [45]	–	8.81	2.35
ResNet (referred from [46])	1.7	6.61	2.01
FractalNet [47]	38.6	5.22	2.01
Wide ResNet [48]	36.5	4.00	–
DenseNet ($k = 40$) [48]	27.2	3.74	1.59
DenseNet + BC ($k = 40$)	25.6	**3.46**	–
EGCM (ConvBlock)	2.23	4.93 (5.05 ± 0.16)	–
	1.80	–	1.63 (1.70 ± 0.09)
EGCM (ShakeShakeBlock)	9.27	3.84 (3.96 ± 0.17)	–
	10.2	–	$\mathbf{1.49}$ (1.58 ± 0.08)

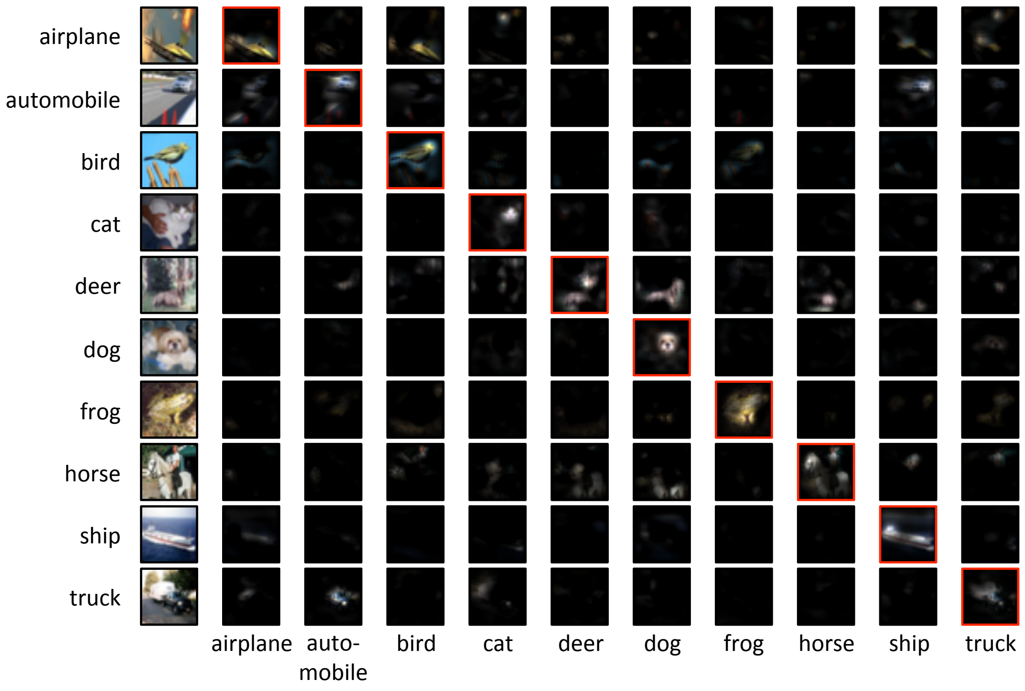


Fig. 3. Examples of visualizations obtained by Evolutionary Generative Contribution Mappings for CIFAR-10.

of epochs and used the maximum validation accuracy of the last five epochs as the fitness value.

After the architecture search process was completed, the best CNN model was retrained for a specific number of epochs using all the training data, including the original training set and additional training set on SVHN. The training settings were similar to those used for the architecture search. In particular, we initialized the weights as described in [50] and used the SGD with a Nesterov momentum of 0.9, mini-batch size of 64, weight decay of 1.0×10^{-4}, and initial learning rate of 0.1. On CIFAR-10, SVHN, and two-digit MNIST, each network was trained for 310, 75, and 75 epochs, respectively. For CIFAR-10, the learning rate was annealed with the cosine annealing schedule [52] with $T_0 = 10$ and $T_{\mathrm{mul}} = 2$, while we used $T_0 = 5$ and $T_{\mathrm{mul}} = 2$ for SVHN and two-digit MNIST [1]. We then recorded the test accuracy at the final epoch.

[1]We did not conducted exclusive hyperparameter tuning for retraining.

C. Search Space and Genetic Algorithm Settings

For the CGP-block configurations, we used standard CGP [53], [54] with rows $N_r = 3$, columns $N_c = 5$ and level-back parameter $L = 3$. On CIFAR-10, SVHN, and two-digit MNIST, the maximum number of CGP-blocks was nine, nine and five, respectively. Every node in the CGP-blocks represents a specific node function. For every intermediate node in the CGP-block, the node function was chosen from the basic function set. For every transition node in the first and the second half of the CGP-blocks, the node function was chosen from the downsampling function set and the upsampling function set, respectively. Throughout this study, the number of channels F and the kernel size k of the convolutional layer were chosen where $F \in \{32, 64, 128\}$ and $k \in \{1, 3, 5\}$.

For the network architecture search, we used the minimal generation gap (MGG) model [41] as the generation

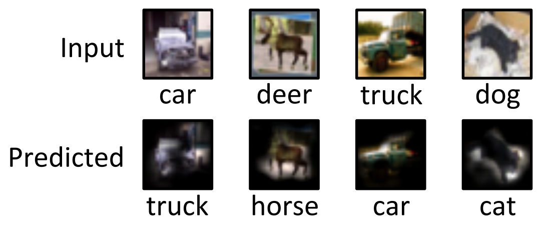


Fig. 4. Examples of visualizations for the misclassified examples. These visualizations explain why the network failed to predict, and make the mistakes seem justifiable.

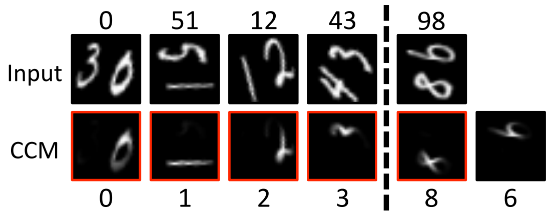


Fig. 6. Examples of the class contribution maps visualizations. Evolutionary Generative Contribution Mappings successfully highlights the second digit.

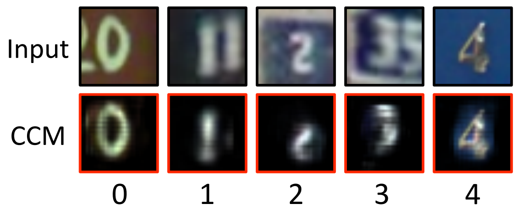


Fig. 5. Examples of visualizations obtained by Evolutionary Generative Contribution Mappings for the Street View House Number dataset together with the original images.

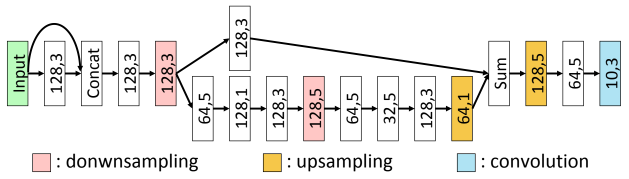


Fig. 7. One of the architectures obtained by our method on CIFAR-10. Each node indicates the number of filters F and the kernel size k in the format (F, k)

alternation model. We chose the number of generations as $N_{\text{gen}} = 250$, population size as $N_{\text{pop}} = 10$, children size as $N_{\text{child}} = 4$, tournament size as $T = 2$, crossover rate as $\gamma = 0.8$, mutation rate as $\mu = 0.05$, and skip probability as $r_{\text{skip}} = 0.2$.

D. Results

1) Comparison with Well-known Convolution Neural Networks: We compare the classification performance of EGCM with well-known CNN models. Here, we emphasize that our goal was *not to achieve a state-of-the-art* performance in terms of image classification, but rather to show that our method maintains high interpretability. In this context, we did not apply any enhancement techniques, such as data normalization and cutout [55], for training the network.

The error rate and the number of parameters are summarized in Table I. As can be seen in this table, EGCM achieves a competitive performance with well-known CNN models. EGCM (ConvBlock) shows a good balance between the performance and the number of parameters, although this performance is not on a par with that of DenseNet [48] and Wide ResNet [48]. In addition, whereas these CNNs use highly functional components, including ResBlock and DenseBlock, our model built only upone standard components (i.e. convolutional layers and skip connection) to achieve this performance. EGCM (ShakeShakeBlock) shows the better performance compared to EGCM (ConvBlock). These models have approximately 10.0M parameters, but outperform most of the models, such as FractalNet [47] and Wide ResNet [48], which have more than 30.0M parameters. This implies that our GA allows to find the appropriate CNN architectures for a given dataset.

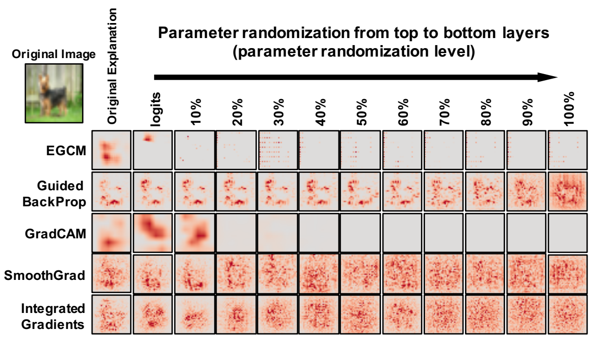
2) Analysis of the Class Contribution Maps: Figure 3 shows class contribution maps (CCM) visualizations for each class of CIFAR-10. In this figure, the class discriminative regions are hightly highlighted; e.g. the ‘airplane’ regions are strongly highlighted but not the ‘sky’ background region. In addition, we can see that EGCM successfully localizes the target objects, even though they are small. This implies that our visualizations are high-resolution enough to capture fine-grained class discriminative detail in the image, which is normally lost in the prevailing convolutional neural network.

The visualizations for the misclassified examples are provided in Figure 4. This visualization explains why the network failed to predict, and makes the mistakes seem justifiable. For instance, in this figure the network misclassifies a deer as a horse; however, the highlighted region in its visualization does look like a horse.

Finally, Figure 5 illustrates a number of CCM visualizations for SVHN. As can be seen in this figure, the CCM visualizations correctly localize the target class number.

3) Two-digit MNIST Results: On two-digit MNIST, EGCM achieved a classification error rate of 1.99%. Intuitively, the second digit should be highlighted in the identification of the second digit. In this sense, the CCM visualizations illustrated in Figure 6 (left) show that EGCM successfully identifies the important region in the image and ignores others. An interesting result is shown on the right-hand side of Figure 6. As seen in this figure, the input image ‘98’ could be misclassified as ‘86’. In this example, the network’s classification is correct; however, we can see that both digits in the image are highlighted in the CCM visualizations. This visualization explains how the network confuses ‘98’ and ‘86’ to reach classification decision.

4) Analysis of the Obtained Architectures: Figure 7 illustrates one of the architectures obtained by our method on CIFAR-10. The obtained architecture is one that has skip connections and various numbers and sizes of filters, which is difficult to design manually. Furthermore, we notice that most of the network architectures do not excessively downsample



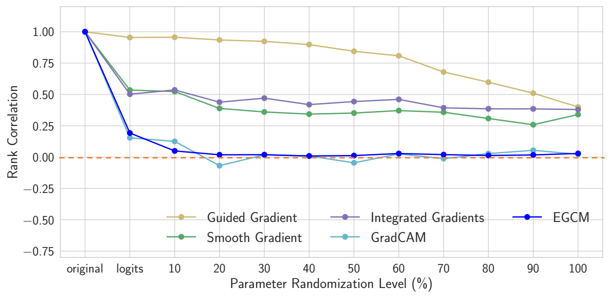
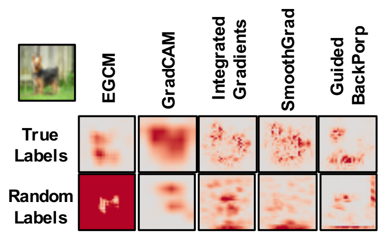


Fig. 8. Effects of parameter randomization level (columns) for the dog. The visualizations from left to rights show complete randomization of network weights. We used the normalized class score for visualization.



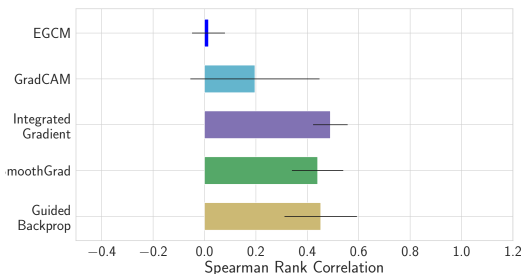


Fig. 9. Visualization for a true model and model trained on random labels.

the inputs. This may imply that a large number of downsampling operations to reduce the spatial size of the feature maps is not efficient in our models. Moreover, throughout these experiments we can observe that most of the network architectures have skip connections and compose modules similar to the Attention module [56]. This suggests that EGCM automatically discovers the promising architectures during an evolution process.

5) Sanity Checks: To evaluate the quality of our visualizations, we performed sanity check [19]. Following [19], we performed two randomization tests, model parameter randomization test and data randomization test. We then measured the similarity of the class contribution maps (saliency) using the Spearman rank correlation. We note that our goal is not to compare our method with all prior explanation methods, but rather to verify the efficiency of our method.

Although Adebayo *et al.* [19] reported that some existing saliency methods showed a visual similarity to the original saliency maps, it was not possible to make out the structure of dog in our visualizations as soon as the top layers were randomized; see Figure 8. This visual similarity is reflected in the Spearman rank correlation (see right-hand side of Figure 8). We show visualizations from the data randomization test and the result of Spearman rank correlation in Figure 9. As can be seen in this figure, our visualizations completely failed to highlight the target objects. This result suggests that our visualizations are sensitive to randomizing labels.

The experimental results show that our visualizations are sensitive to both model parameters and data. This indicates that our method passes the sanity checks, and our visualization successfully explain the relationship between inputs and outputs.

V. Conclusion

In this paper, we proposed a novel technique, called *Evolutionary Generative Contribution Mappings* (EGCM) for achieving a high classification performance together with high interpretability. We empirically showed that EGCM achieves a competitive classification, maintaining high interpretability for classification decisions. However, we achieved this by using a small number of population, children, and generations. This suggests that there is more room for improvement and expansion of our GA.

In the future, we plan to apply our method to larger datasets, and extend its possible applications. Owing to its computational efficiency, we need to adapt some techniques, such as early termination, which are often used in the evolutionary computation community, to accelerate the architecture search. Another approach is to follow the idea in [34], and search for transferable architectures. This is one of the important future research directions.

Acknowledgment

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.
[3] D. Castelvecchi, "Can we open the black box of ai?" *Nature*, vol. 538, no. 7623, pp. 20–23, 2016.
[4] Z. C. Lipton, "The mythos of model interpretability," *arXiv preprint arXiv:1606.03490*, 2016.
[5] L. Breiman, "Statistical modeling: The two cultures," *Statistical Science*, vol. 16, no. 3, pp. 199–231, 2001.
[6] S. Sarkar, T. Weyde, A. S. d'Avila Garcez, G. G. Slabaugh, S. Dragicevic, and C. Percy, "Accuracy and interpretability trade-offs in machine learning applied to safer gambling," in *NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic aApproaches*, 2016.
[7] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *ICCV*, 2017, pp. 618–626.
[8] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.
[9] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, pp. 66–72, 1992.
[10] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," University of Montreal, Tech. Rep., 2009.
[11] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.
[12] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," vol. arXiv preprint arXiv:1703.01365, 2017.
[13] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *ICML*, 2017, pp. 3145–3153.
[14] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *ICCV*, 2017, pp. 3429–3437.
[15] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *NIPS*, 2017, pp. 6967–6976.
[16] P.-J. Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne, "Learning how to explain neural networks: PatternNet and PatternAttribution," in *ICLR*, 2018.
[17] J. Chen, L. Song, M. Wainwright, and M. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," in *ICML*, 2018, pp. 883–892.
[18] A. Chattopadhyay, P. Manupriya, A. Sarkar, and V. N. Balasubramanian, "Neural network attributions: A causal perspective," in *ICML*, 2019, pp. 981–990.
[19] J. Adebayo, J. Gilmer, M. Muelly, I. J. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *NIPS*, 2018, pp. 9505–9515.
[20] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *CVPR*, 2016, pp. 2921–2929.
[21] S. Arai and T. Nagao, "Intuitive visualization method for image classification using convolutional neural networks," *Information Processing Society of Japan. Transactions on mathematical modeling and its applications*, vol. 10, no. 2, pp. 1–13 (in Japanese), 2017.
[22] A. Radhakrishnan, C. Durham, A. Soylemezoglu, and C. Uhler, "Patchnet: Interpretable neural networks for image classification," in *NIPS Workshop*, 2018.
[23] O. Li, H. Liu, C. Chen, and C. Rudin, "Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions," in *AAAI*, 2018, pp. 3530–3537.
[24] D. Alvarez Melis and T. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *NIPS*, 2018, pp. 7775–7784.
[25] R. Hu, J. Andreas, T. Darrell, and K. Saenko, "Explainable neural computation via stack neural module networks," in *ECCV*, 2018.
[26] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.
[27] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018, pp. 4095–4104.
[28] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *ICGA*, 1989, pp. 379–384.
[29] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 1–37.
[30] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.
[31] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *ICML*, 2017.
[32] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *GECCO*, 2017, pp. 497–504.
[33] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2017.
[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018, pp. 8697–8710.
[35] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *AAAI*, 2018, pp. 2787–2794.
[36] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018.
[37] M. Suganuma, M. Ozay, and T. Okatani, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search," in *ICML*, 2018.
[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.
[39] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814.
[40] X. Gastaldi, "Shake-shake regularization," *arXiv preprint arXiv:1705.07485*, 2017.
[41] H. Satoh, I. Ono, and S. Kobayashi, "Minimal generation gap model for gas considering both exploration and exploitation," in *Proceedings of IIZUKA '96*, 1996, pp. 494–497.
[42] A. Krizhevsky and G. E. Hinton, "Learning multiple layers of features from tiny images," *Technical report*, 2009.
[43] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
[44] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *ICML*, 2013, pp. 1319–1327.
[45] M. Lin, Q. Chen, and S. Yan, "Network in network," in *ICLR*, 2014.
[46] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *ECCV*, 2016, pp. 646–661.
[47] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," in *ICLR*, 2017.
[48] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016, pp. 87.1–87.12.
[49] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 4700–4708.
[50] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015, pp. 1026–1034.
[51] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *ICML*, 2013, pp. 1139–1147.
[52] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with restarts," in *ICLR*, 2017.
[53] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *EuroGP*, 2000, pp. 121–132.
[54] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.
[55] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1805.09501*, 2017.
[56] W. Chen, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *CVPR*, 2017, pp. 3156–3164.