A Genetic Algorithm for Convolutional Network Structure Optimization for Concrete Crack Detection

Spencer Gibb, Hung Manh La, and Sushil Louis

***Abstract*— A genetic algorithm (GA), is used to optimize the many parameters of a convolutional neural network (CNN) that control the structure of the network. CNNs are used in image classification problems where it is necessary to generate feature descriptors to discern between image classes. Because of the deep representation of image data that CNNs are capable of generating, they are increasingly popular in research and industry applications. With the increasing number of use cases for CNNs, more and more time is being spent to come up with optimal CNN structures for different applications. Where one CNN might succeed at classification, another can fail. As a result, it is desirable to more easily find an optimal CNN structure to increase classification accuracy. In the proposed method, a GA is used to evolve the parameters that influence the structure of a CNN. The GA compares CNNs by training them on images of concrete containing cracks. The best CNN after several generations of the GA is then compared to the state-of-the-art CNN for crack detection. This work shows that it is possible to generalize the process of optimizing a CNN for image classification through the use of a GA.**

I. Background and Related Work

In the proposed method, a GA is used to optimize the parameters of a CNN. The goal of this method is to optimize the performance of the CNN at detecting cracks in concrete surface. Concrete crack detection is a type of image classification problem where classic image analysis methods have failed due to the various difficulties in the crack detection process, including: illumination changes, background clutter in images, and the shape of the cracks. CNNs have shown that they are capable of attaining high accuracy at crack detection in [1], but choosing a CNN structure that performs well is a time consuming process due to the number of parameters that affect the network structure. For this reason, a GA is applied to search the space of possible network structures. In this section, a review of general image classification is provided, as well as an explanation of crack detection and CNNs. Related work is also provided to help show the novelty of the method.

Financial support for this study was provided by the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center (http://inspire-utc.mst.edu) at Missouri University of Science and Technology. The views, opinions, findings and conclusions reflected in this publication are solely those of the authors and do not represent the official policy or position of the USDOT/OST-R, or any State or other entity. Dr. Louis was supported by grant number N00014-17-1-2558 and N00014-15-1-2015 from the Office of Naval Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

Spencer Gibb and Dr. Hung La are with the Advanced Robotics and Automation (ARA) Laboratory. Dr. Sushil Louis is with the Evolutionary Computing Systems Lab (ECSL), Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557, USA. Corresponding author: Hung La, email: hla@unr.edu

A. Image Classification

Image classification is used in many applications including face recognition, object detection, and scene recognition [2], [3], [4]. Typically, the image classification process involves modeling a set of training data so that it is possible to discern between two or more classes of objects or images using a classifier. Then, once the data is modeled, the classifier can be applied to a new set of data in an application setting. One example of a real-world image classification task is crack detection in images of concrete [5], [6]. Locating cracks in images of concrete can be useful in applications such as civil infrastructure inspection, where it is desirable to be able to perform inspection of parking garages, bridges, and other structures in a fully autonomous manner [7], [8], [9]. Using the location of cracks found in images from an on-board camera, it is possible to offer a convenient and cost-effective map of the area.

Crack detection in concrete and pavement has been attempted with many image processing, computer vision, and machine learning techniques. Initial attempts to detect cracks in images of concrete were performed using methods such as the Canny edge detector, Sobel filter, Laplace of Gaussian (LoG) [10], [11] for edge detection, and other simple methods. These methods do not perform well in cases where the images are noisy, have shadows, or have textured backgrounds. In an attempt to improve crack detection accuracy compared to these classic methods, newer modeling and learning techniques have been applied to crack detection in recent research. Among these techniques are random forest classifiers, neural networks, percolation-based edge detection, frequency domain filtering, and CNNs [12], [13], [14], [15], [1]. While modern work tends to show high accuracy in crack classification, various models and techniques work better or worse depending on the test environment, camera system, lighting, the presence of other objects in the image, and the size and depth of cracks in the test images [16].

Some of our previous work has focused on implementing the CNN in [1] on a robotic platform designed at the University of Nevada, Reno campus. This network is the best crack detection method, prior to the proposed method, that we are currently aware of and is capable of being run in quasi-real time. However, because of the conditions discussed previously and the difficulty of the crack detection problem, this network is still not capable of detecting cracks reliably under difficult circumstances. For these reasons, a more

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

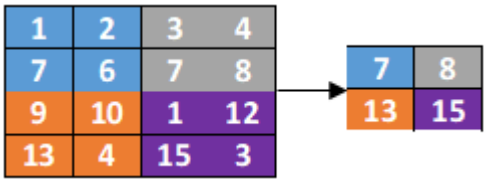

Fig. 1: An example of max-pooling. The input (left) is split into four parts and each part is searched for the maximum local value. The output (right) contains the maximum values found in the search.

robust solution to the crack detection problem is desired. While deep learning - and more specifically CNNs - provide the best results of any current learning method, constructing an optimal network is an extremely time consuming task due to the size of the search space, which requires a brief introduction to CNNs in order to understand.

B. Convolutional Network Structure

There are several types of layers that make up the structure of CNNs including: convolution layers, pooling layers, and fully-connected layers. Convolution layers perform multiplication between the input image, or two-dimensional array, and a filter that is also a two-dimensional array. This multiplication happens at every location in the input array and the result is then summed. This process yields another two-dimensional array and can be described mathematically as follows:

$$f[x,y] * g[x,y] = \sum_{i=-\frac{s}{2}}^{\frac{s}{2}} \sum_{j=-\frac{s}{2}}^{\frac{s}{2}} f[i,j] \cdot g[x-i, y-j], \quad (1)$$

where f is the input image, or array, g is the filter, s is the size of the filter, which is a square filter, and x and y are locations in the input arrays. However, Equation (1) typically happens many times in each convolution layer but with different filters used for g. Thus, there can be a set of filters used in each convolution layer called $g_1, g_2, ..., g_f$, where f is the number of filters in each convolution layer. With a filter size of s and l convolution layers in the network, there are at least $s^2 \cdot f \cdot l$ weights that need to be optimized during the training process, which can quickly create a problem that is not feasible depending on the application requirements.

The second major type of layer in a CNN is a pooling layer. There are several types of pooling layers, but the one used most frequently because of its efficacy is the max-pooling layer, which performs sub-sampling of its input through selecting the maximum value in each region and only passing that value on. An example of this process can be seen in Figure 1. Max-pooling is performed to reduce the complexity of the output throughout the CNN, since it sub-samples and produces a lower dimensional output. However, information is not lost here as maximum values are capable of preserving the important information throughout the convolution process.

The last type of layer that is used often in CNNs is a fully-connected layer. This type of layer works in a similar way to a multilayer perceptron network, which is a classic type of neural network. In fully-connected layers, each input node is connected to each output node directly, and a weight is associated with each connection. These weights are optimized using the back-propagation process [17], which will not be discussed here for the sake of space; however, an output node in the fully-connected layer sums its value as follows:

$$Output = \omega_1 V_1 + ... + \omega_N V_N, \quad (2)$$

where there are N nodes connected to the output node in Equation (2), and each node has an associated weight, ω_i, and value, V_i, where $i = 1, ..., N$. Fully-connected layers are typically connected to the end of a CNN as a final way to introduce non-linearity into the network. However, large fully-connected layers need to be used with caution since they can quickly introduce many weights to be optimized and greatly increase the amount of training time required for the network. This is a result of the N^2 weights that need to be optimized for each fully-connected layer.

C. Application of a Genetic Algorithm

CNNs require a large number of parameters to be optimized function at peak performance, which can lead to long training times for networks and often require a deep understanding of the application and data in order to be used successfully. The list of parameters that must be considered in order to construct a CNN include: the number of layers in the network, the activation functions used in the network, the size of layer operations (for convolution and pooling/sub-sampling), the number of layers used in each convolution, the training parameters for the optimizer, and more. This results in an infinite number of possible network structures, and it is often too time consuming to get good results for image classification problems because of the complexity of constructing deep learning networks. To illustrate this point, the network structure used in [1] is shown in Figure 2. In Figure 2, there are three convolution layers, two pooling/sub-sampling layers, and two activation layers. In this case, the parameters that had to be determined in order to construct this network included convolution layer sizes, pooling layer sizes, activation function types, network depth, optimization function type, and learning rate of the optimization function. The purpose of the proposed method is to use a GA to come up with a highly accurate CNN through optimizing the network parameters, which will help to eliminate some of the difficulty in constructing an accurate network. Using the network in [1] as a baseline, it will be possible to validate the performance of newly constructed networks against the state of the art method.

The remainder of the paper is structured as follows. In Section II, a brief review of the previous literature in this area of research is provided. In Section III, an in-depth description of the network optimization process is discussed. Then, in Section IV, experimental results are provided along with their

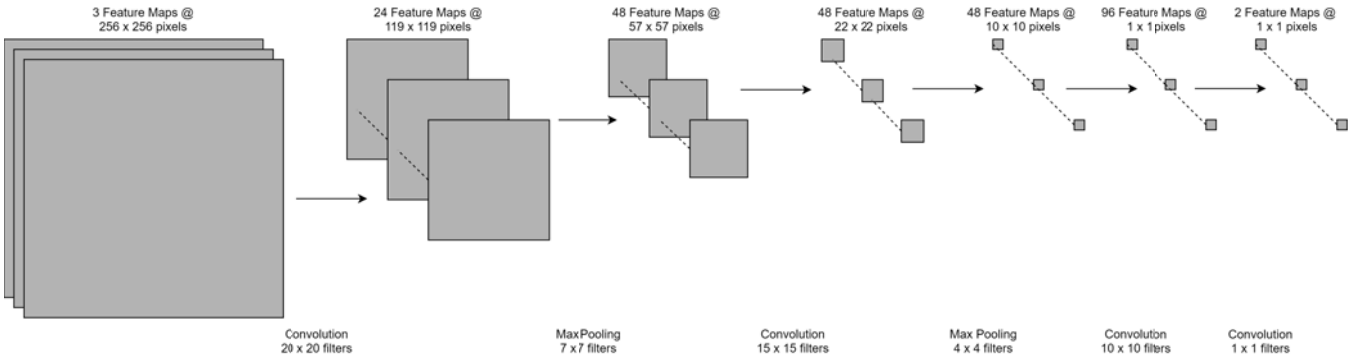

Fig. 2: The network structure from [1] used as the baseline for the proposed method. The ReLU activation function is used in all layers in this network.

explanation and insights. Finally, a conclusion is offered in Section V to summarize the proposed method.

D. Related Work

There are three types of existing research related to using GAs to improve the results of CNNs: hyper parameter optimization, structure optimization, and loss function optimization. Hyper parameter optimization consists of using GAs to optimize parameters such as the learning rate of the optimization function at the end of a CNN, the number of layers used in convolution, and the type of optimization function used given a set of predefined functions [18], [19], [20]. Generally, hyper parameter optimization is simple because the search space is somewhat small and an exhaustive search may be possible given the range of each parameter and the training time for the CNN. Structure optimization, as performed in [21], consists of using a GA to optimize the size of the convolutional layers being used in the network. Unfortunately, this is performed on an extremely simple network, with small images, and on a problem that is already solved on the data set used for testing. While an exhaustive search may not be reasonable in the case of the network in [21], it would be reasonable to place further constraints on the problem to achieve a globally optimal solution without much effort. In addition, the data set used in [21] is a simple face recognition task for modern classification methods, as a classification accuracy of 96% was achieved by a method developed in 1991. Finally, loss function optimization is performed in [22], where an existing loss function is optimized using a GA. This type of optimization does not directly affect the structure or function of the CNN, but instead optimizes an added classification layer.

II. Methodology

As discussed in the previous section, there are several ways to apply GAs to optimize components of CNNs. In this research, a GA has been applied to a variable depth CNN. This means that in the proposed method, the GA evolves the network depth, the layer size, and the hyper parameters of the network. The size of layers in the CNN affects the level of details that are recognized by the CNN, while the hyper parameters control the number of representations of the image throughout the convolution process, which directly affects overall accuracy of the network. This combination guarantees that an exhaustive search is not feasible, which is a case where GAs have excelled previously. In this section, an overview of the methodology proposed here is given. An overall diagram showing the methodology of the proposed method is shown in Figure 3.

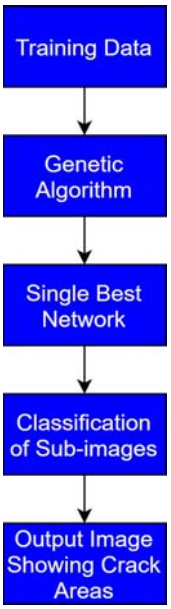

Fig. 3: A block diagram showing the overall approach, from the training data to the output image showing where cracks were detected.

A. Network Structure

In order for the GA to have as much control as possible over the network structure, it is necessary to allow for many types of networks through the evolution process. The network structure parameters can be seen in Table I. Each individual in the population consists of a chromosome represented, in binary, by 14 bits total. While $2^{14} \approx 16,000$, it is important to note that an exhaustive search is not feasible. Some of the networks take more than one hour to train, which would result in a total of nearly 2 years of training time on a

Parameter	Convolution Layers	Max-pooling Layers	Size of Conv. Filters	Number of Conv. Filters
Value Range	1 - 16	1 - 4	1 - 32 (pixels)	1 - 32
Bit Representation	4 bits	0 bits	5 bits	5 bits

TABLE I: An overview of the main parameters that can be evolved by the GA.

Fig. 4: Images from the "crack" training data (top), and images from the "no-crack" training data (bottom).

single NVIDIA 1080 GTX graphics processing unit (GPU). The variety of possible network architectures reinforce the fact that this problem is non-trivial and time consuming.

Each time an evaluation of an individual takes place, a network must first be constructed based on the individual's chromosome. The structure of the network depends largely on the number of convolution layers specified in the chromosome. A max-pooling layer is added to the network after every set of four convolution layers to reduce the complexity of the network. The network always ends with a fully-connected layer with two nodes, which uses a softmax activation function. The final fully-connected layer will output probabilities that the input image belongs to the "crack" class and the "no-crack" class, respectively. Everywhere else, the rectified linear unit (ReLU) activation function is used. Each pooling layer uses a 4 by 4 filter with a stride of 4. Stochastic gradient descent (SGD) is used as the optimization method, with a learning rate of 0.001. SGD is an optimization technique that is used in the back-propogation process to optimize the weights within the network. SGD is described mathematically as:

$$\theta = \theta - \alpha \cdot \nabla_\theta \cdot J(\theta; x^{(i)}, y^{(i)}), \tag{3}$$

where θ is the parameter being optimized, α is the learning rate, J is the objective function, $x^{(i)}$ is sample i, and $y^{(i)}$ is label i.

Once a model is constructed using a chromosome, the model is trained on a set of training data. The training data set has been labeled manually as either belonging to the "crack class" or to the "no-crack" class. Examples of images from both classes can be seen in Figure 4. The training data consists of 3000 images: 1500 crack images, and 1500 no-crack images. For comparison, the network in [1] utilizes 60,000 images in training. Each network is then given 30 epochs of training time to back-propagate and the epoch displaying the highest training accuracy is kept, so that any decrease in network performance or errors will not be accidentally incorporated into the evolution process. The network construction process is outlined in Algorithm 1.

Algorithm 1: NETWORK CONSTRUCTION BASED ON CHROMOSOME

Input: l = number of convolution layers
s = convolution filter size
f = convolution filter count
Output: net = network containing the structure described by the chromosome

```
conv_sections = int(l ÷ 4)
remainder = f%l
net = new Network()
if conv_sections > 0 then
    for i ← 0 to conv_sections do
        for j ← 0 to 4 do
            net.add_layer(type = "convolution", size = s, filters = f)
        net.add_layer(type = "pooling")
    if remainder > 0 then
        for i ← 0 to remainder do
            net.add_layer(type = "convolution", size = s, filters = f)
        net.add_layer(type = "pooling")
else
    for i ← 0 to remainder do
        net.add_layer(type = "convolution", size = s, filters = f)
    net.add_layer(type = "pooling")
net.add_layer(type = "fully − connected", size = 2)
```

B. Fitness Function

After a network is constructed based on an individual's chromosome, it is necessary to evaluate the fitness of that individual for the selection and crossover processes later. The fitness function used in the proposed method is a simple accuracy metric. Each individual, and its corresponding network, is evaluated based on the accuracy that it has when classifying a set of test images. The test set contains 600 images: 300 images of cracks, and 300 images with no cracks. The fitness of a network is a decimal value between 0 and 1, and can be described mathematically as:

$$f = \frac{c}{t}, \tag{4}$$

where c is correct classifications, t is total classifications, and f is fitness.

Equation (4) was also applied to the base network found in [1], which returns a value of 0.8017, indicating that the base network correctly classified $\approx 80\%$ of the test data correctly after being trained on the 3000 training images.

Algorithm 2: EVOLUTIONARY STRATEGY (μ, λ)

Input: $p =$ population of individuals
$\mu =$ individuals to select for next generation
$\lambda =$ children to produce prior to selection
$g =$ number of generations
$cp =$ crossover probability
$mp =$ mutation probability
Output: $p =$ the input population, evolved in place

```
p_size = length(p)
for i ← 0 to p_size do
    evaluate(p[i])
offspring = new Population[λ]
for i ← 0 to g do
    for j ← 0 to λ do
        r = rand([0, 1])
        if r < cp then
            /* Generate an offspring using
               crossover.                    */
            offspring[j] = crossover(p)
        else if r < cp + mp then
            /* Generate an offspring using
               mutation.                     */
            offspring[j] = mutation(p)
        else
            /* Choose an offspring from
               parents randomly.             */
            offspring[j] = choice(p)
    for i ← 0 to λ do
        evaluate(offspring[i])
    /* Select μ individuals for the
       next generation.                      */
    p = selection(p, offspring)
```

C. Genetic Algorithm

The proposed method utilizes the (μ, λ) algorithm, which is detailed in Algorithm 2. Given a population of size μ, this evolutionary strategy utilizes either crossover, mutation, or a random choice to generate λ children. Of the total available $\mu + \lambda$ individuals, μ are selected to carry on to the next generation. Selection is performed using fitness proportional selection, meaning that individuals with a higher fitness are more likely to carry on to the next generation than those with lower fitness values. This algorithm was chosen because it has been shown that the canonical GA is not a good function optimizer [23]. In addition, this algorithm offered the ability to have a wider choice of individuals when generating a new generation, since it is entirely possible to keep the parent population if the offspring prove to perform poorly. The parameters for this algorithm can be found in Table II. A brief review of the crossover, mutation, and random choice functionality has also been provided.

D. Crossover, Mutation, and Random Choice

For the (μ, λ) algorithm, one point crossover, bit flipping mutation, and random choice picking are all used to generate offspring. One point crossover was used in the proposed method. In one point crossover, children are generated through picking a single point in the bit strings of the parent. Then the first child is formed from taking everything left of that point from the first parent and everything right of that point from the second parent. The second child is generated by doing the reverse. Mutation is performed by bit flipping. An offspring is generated through mutation in this case by selecting one parent from the parent population at random. Then, a random number is generated for each bit in the bit string. If the random number, r, is below the probability that each bit has to be flipped, then the bit is flipped. Otherwise, the bit remains the same. Finally, if crossover or mutation are not performed, then an offspring must still be generated. In this case, the offspring is generated by randomly selecting a parent from the parent population. More information on the crossover operation, mutation operation, and overall genetic algorithm can be found in [24].

E. Software and Implementation

Several free software packages were used in the implementation of the proposed method. For the GA implementation, a Python package called DEAP was utilized [25]. Each network was constructed using the bit strings generated by DEAP and a package called Keras, which is a Python package for intuitive neural network creation [26]. Because of the time required to train each network, this project required several GPUs to be used. The evaluation of offspring was distributed accross multiple machines using network communication over secure shell (SSH). This allowed for a static allocation of evaluation tasks between five machines running Nvidia 1080 GTX GPUs and greatly decreased the overall runtime of the method.

III. EXPERIMENTAL RESULTS

The GA was run with a population size of 30 for 10 generations. The plot for this can be seen in Figure 5. This graph shows the maximum, average, and minimum fitness of each generation across all 10 generations. Maximum fitness is the fitness of the best network in the population, the average fitness is the mean fitness of the population, and the minimum fitness is the fitness of the lowest performing network in the population. It can be seen from the graph that the proposed method surpasses the performance of the network in [1] by the third generation. While the population converges by generation 7, there is still some diversity between the networks, in that they have varying numbers of convolution layers, varying numbers of pooling layers, different filter sizes used in the convolution layers, and varying numbers of filters used in each convolution layer. By

Population size	Generations	Selection	Crossover	Probability of crossover	Mutation	Probability of mutation
30	10	Fitness proportional	One point	0.67	Bit flip	0.03

TABLE II: The parameters of the (μ, λ) algorithm used to evolve the structure of CNNs.

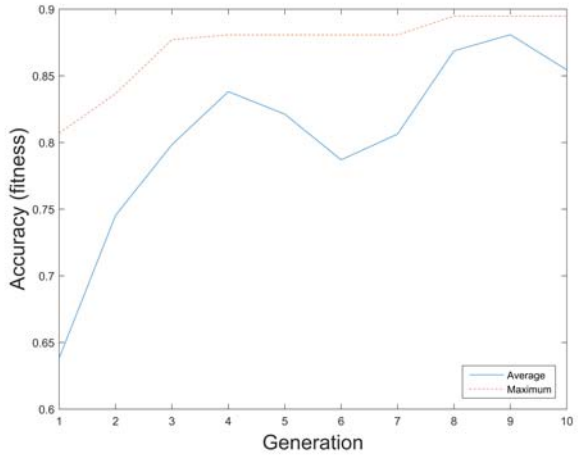

Fig. 5: The performance of the GA across 10 generations with a population size of 30 individuals. This algorithm was run 10 times and the data in this graph represents average values accross those runs. The best network achieved through the GA obtains 89.17% accuracy, which can be seen in generation 8.

Parameter	Value
Number of Convolution Layers	11
Number of Filters Per Layer	20
Size of Filters	15 by 15 (pixels)

TABLE III: The parameters of the best network obtained from the GA.

the generation 9, the maximum fitness increases to 89.17%, which is 9% higher than the network in [1], which achieved a fitness of 80.17% after being trained on the same training data. A table showing the parameters of the best network from the GA can be seen in Table III.

To provide a visual example of the performance difference between the proposed method and the network in [1], the best network from the GA was used on an image containing cracks. The network in [1] was tested on the same image. The networks classified sub-images within the input image, since the training process was performed using 256 by 256 pixel images. The input image was split into 256 by 256 pixel sub-images and each sub-image was provided to the networks. If a network classified the input sub-image as a "crack" image, then the output image contains that sub-image, but if the network classified the input sub-image as a "no-crack" image, then the output image has had that section removed. This means there were no cracks detected where the output images are black. The original image and output images can be seen in Figure 6. The original image was chosen because it showcases several aspects that make crack classification difficult, including: dark spots in the background that look

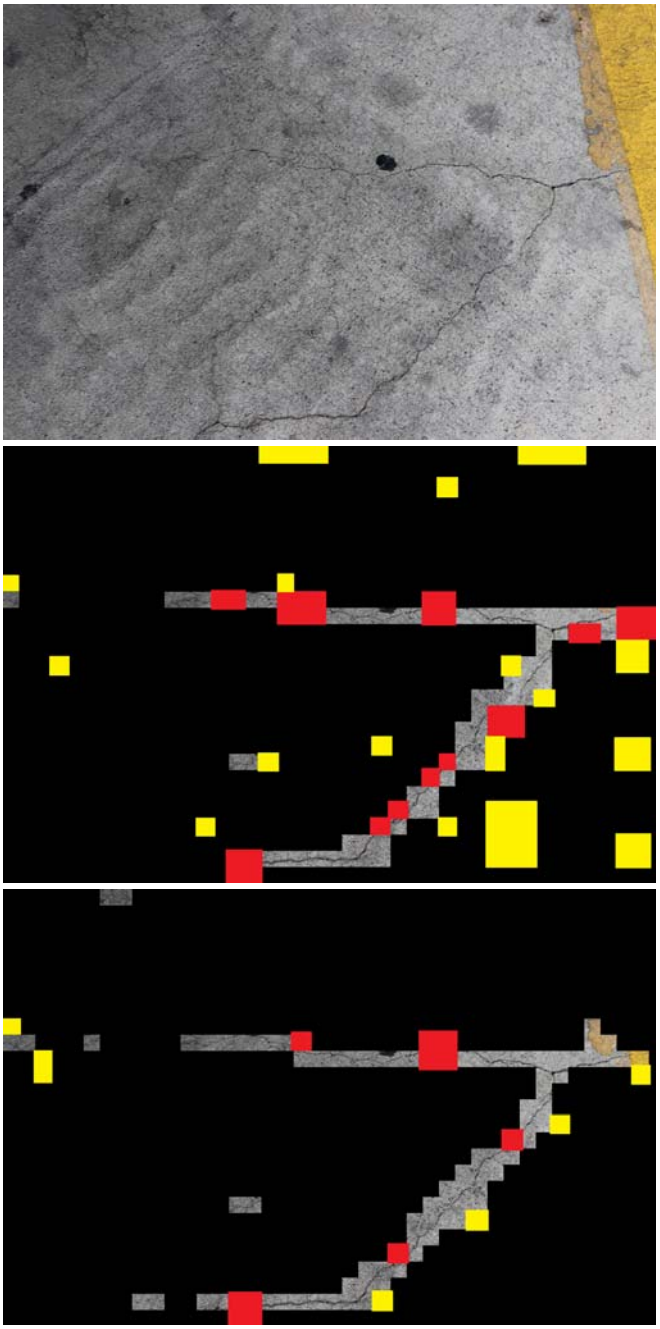

Fig. 6: A visual example of the performance of the network generated by the genetic algorithm. The original image (top), the performance of the network in [1] (middle), and the performance of the network generated with the proposed method (bottom). False positives have been covered by light gray (filled) rectangles and false negatives have been coviered by dark gray (filled) rectangles.

similar to oil and can be easily mistaken for cracks, yellow material that the networks have not been exposed to, and

Network	Correct Classifications	Incorrect Classifications	Accuracy	False Positives	False Negatives
Network from [1]	4335	545	0.888	481	64
Best Network from GA	4586	294	0.94	280	14

TABLE IV: The results from both networks after classifying 10 test images split into 256 pixel by 256 pixel sub-images. The total number of sub-images classified was 4880.

difficult lighting conditions. This classification process and the process of removing the "no-crack" sections from the output image are shown as the final two steps in Figure 3.

It can be seen in Figure 6 that the best network from the GA performs better than the network from [1] in two major ways: lack of false positives and crack continuity. The middle image in Figure 6 contains several positives in the bottom right corner of the image where cracks were detected, but there are none actually present in the original image. On the other hand, there are no such false positives in the bottom right corner of the bottom image in Figure 6. Cracks are typically not straight, which can be seen in Figure 6. Because of this, certain sections of cracks may be straight and then suddenly diverge from their path. The best network from the GA does an adequate job of detecting different regions of the same crack that may not appear continuous, but are in fact, whereas the network from [1] classifies these regions as "no-crack" regions and loses the crack continuity.

In addition to the visual example provided in Figure 6, a larger comparison of results is provided in Table IV. Ten images of concrete surface were split into 256 pixel by 256 pixel sub-images, which resulted in a total of 4880 images. These images were classified by each network as being either "crack" images or "no-crack" images. False positives are images that do not contain any cracks, but were classified as "crack" images by the classifier. False negatives are images that do contain cracks, but were classified as "no-crack" images by the classifier. In this test the best network from the GA performs approximately 5% better than the previous best network. It is also important to note that the best network from the GA has 46% less false positives than the previous best network and 76% less false negatives.

A. Insights

There are several valuable insights, both intuitive and not, that can be gained from the evolution of CNN structures by a GA. First, deeper networks are not always better. While adding more layers can introduce more non-linearity to the features being generated, if too many pooling layers are introduced, it becomes more difficult for the features to be separable because they consist of less elements. For this reason, networks with 7 to 11 convolution layers performed better than networks with 16 convolution layers, which was the maximum allowed by the implementation. Second, more filters is almost always better than less filters because a higher number of filters produce a deeper representation of features in images. This is intuitive, and reinforced by the fact that all of the top networks produced by the GA used more than 20 filters in each convolution layer. Third, the number of network structures that had to be constructed by the GA to find the best network was only 300, whereas the total number of networks that need to be constructed to perform an exhaustive search is $\approx 16,000$. Lastly, odd filter sizes tend to perform better than even filter sizes. This is a result of the convolution operation, which needs to have the filter centered at a pixel in the input image. All of the best performing networks used odd filter sizes.

IV. Conclusion and Future Work

The proposed method utilizes a GA for CNN structure optimization. This method evolved several parameters that dictate the structure of a CNN, including: number of convolution layers, size of convolution filters, and number of convolution filters used in each layer. Evolving CNN structures produces high-performance networks that have higher classification accuracy than the state-of-the-art network when tested on images of concrete containing cracks. This process allows for a small training set, since mass training data is sometimes difficult to obtain in real-world applications. The process also performs the search for network structures automatically, which removes the need for a deep knowledge of the features being described and the neural network design process. Visual results indicate that the generated networks perform well at crack detection in concrete images and accurately capture the unpredictable nature of crack structure. Future work will include generalization of the method as it is applied to multiple types of image data, as well as more investigation of possible network structures parameters that could increase performance on varying image classification problems. This process would require changing the parameter ranges that the GA evolves. If significantly larger or smaller images are used for training, then it might be beneficial to vary the range of filter sizes. If more computing power is available, then it may be applicable to increase the range of filter counts for each convolution layer. Finally, it may be desirable to allow the network to perform different combinations of layers, so that pooling layers are added to the network more or less often. This process would require some experimentation to find complete generalization of the network optimization process.

References

[1] Y.-J. Cha, W. Choi, and O. Bykztrk, "Deep learning-based crack damage detection using convolutional neural networks," *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, no. 5, pp. 361–378, 2017. [Online]. Available: http://dx.doi.org/10.1111/mice.12263

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, Jul 1997.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–511–I–518 vol.1.

[4] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: An astounding baseline for recognition," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2014, pp. 512–519.

[5] H. M. La, N. Gucunski, S.-H. Kee, and L. V. Nguyen, "Data analysis and visualization for the bridge deck inspection and evaluation robotic system," *Visualization in Engineering*, vol. 3, no. 1, p. 6, Feb 2015. [Online]. Available: https://doi.org/10.1186/s40327-015-0017-3

[6] H. M. La, N. Gucunski, S.-H. Kee, J. Yi, T. Senlet, and L. Nguyen, "Autonomous robotic system for bridge deck data collection and analysis," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 1950–1955.

[7] S. Gibb and H. M. La, *Automated Rebar Detection for Ground-Penetrating Radar*. Advances in Visual Computing: 12th International Symposium, ISVC 2016, Las Vegas, NV, USA, December 12-14, 2016, Proceedings, Part I: Springer International Publishing, 2016, pp. 815–824.

[8] T. D. Le, S. Gibb, N. H. Pham, H. M. La, L. Falk, and T. Berendsen, "Autonomous robotic system using non-destructive evaluation methods for bridge deck inspection," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017.

[9] H. M. La, N. Gucunski, K. Dana, and S.-H. Kee, "Development of an autonomous bridge deck inspection robotic system," *Journal of Field Robotics*, 2017.

[10] R. S. Lim, H. M. La, and W. Sheng, "A robotic crack inspection and mapping system for bridge deck maintenance," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 367–378, April 2014.

[11] R. S. Lim, H. M. La, Z. Shan, and W. Sheng, "Developing a crack inspection robot for bridge maintenance," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 6288–6293.

[12] P. Prasanna, K. J. Dana, N. Gucunski, B. B. Basily, H. M. La, R. S. Lim, and H. Parvardeh, "Automated crack detection on concrete bridges," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 591–599, April 2016.

[13] T. H. Dinh, Q. P. Ha, and H. M. La, "Computer vision-based method for concrete crack detection," in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Nov 2016, pp. 1–6.

[14] T. Yamaguchi and S. Hashimoto, "Improved percolation-based method for crack detection in concrete surface images," in *2008 19th International Conference on Pattern Recognition*, Dec 2008, pp. 1–4.

[15] S. Dorafshan, "Comparing automated image-based crack detection techniques in spatial and frequency domains," 03 2017.

[16] S. Dorafshan, M. Maguire, N. V. Hoffer, and C. Coopmans, "Challenges in bridge inspection using small unmanned aerial systems: Results and lessons learned," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2017, pp. 1722–1730.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451

[18] R. Oullette, M. Browne, and K. Hirasawa, "Genetic algorithm optimization of a convolutional neural network for autonomous crack detection," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, June 2004, pp. 516–521 Vol.1.

[19] S. Fujino, N. Mori, and K. Matsumoto, "Deep convolutional networks for human sketches by means of the evolutionary deep learning," in *2017 Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS)*, June 2017, pp. 1–5.

[20] Y. Kanada, "Optimizing neural-network learning rate by using a genetic algorithm with per-epoch mutations," in *2016 International Joint Conference on Neural Networks (IJCNN)*, July 2016, pp. 1472–1479.

[21] A. Rikhtegar, M. Pooyan, and M. T. Manzuri-Shalmani, "Genetic algorithm-optimised structure of convolutional neural network for face recognition applications," *IET Computer Vision*, vol. 10, no. 6, pp. 559–566, 2016.

[22] V. Ayumi, L. M. R. Rere, M. I. Fanany, and A. M. Arymurthy, "Optimization of convolutional neural network using microcanonical annealing algorithm," in *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, Oct 2016, pp. 506–511.

[23] K. A. De Jong, "Genetic algorithms are not function optimizers," *Foundations of genetic algorithms*, vol. 2, pp. 5–17, 1993.

[24] D. E. Goldberg *et al.*, "Genetic algorithms in search optimization and machine learning," 1989.

[25] "DEAP," https://github.com/DEAP/deap, accessed: 2017-09-30.

[26] "Keras," https://keras.io/, accessed: 2017-09-30.