International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 9, No.2 (Mar-2020)
http://dx.doi.org/10.12785/ijcds/090212

Test Case Generation for Convolutional Neural Network

Hyung Ho Kim[1]

[1] *SolutionLink, South Korea*

Received 27 Sep. 2019, Revised 24 Feb. 2020, Accepted 26 Feb. 2020, Published 01 Mar. 2020

Abstract: In this paper, we present a test image generation approach for Convolutional Neural Network (CNN) that is widely used for image recognition. The goal of our approach has to generate an image satisfying the following two conditions. First, this image activates a specific cell on CNN for checking the effect of this cell. Second, this image looks like a real image as a plausible test case. For this purpose, we combine the activation maximization technique with GAN (generative adversarial network). Even though quite a large number of sample data are used for training, it is infeasible to activate every cell and check its effect. Thus, this technique is useful for verifying cells uncovered in training and, thus, for improving the quality of CNN. To our best knowledge, this is the first attempt to improve the quality of images using the generative modeling approach. With the famous MNIST example, we illustrate the details and benefits of the proposed approach.

Keywords: Deep Learning, Convolutional Neural Network, AI Testing, AI Safety, Test Case Design

1. Introduction

Nowadays, deep learning technologies become widely used for various critical applications such as autonomous driving or optimization of infrastructures. Thus, it is crucial to remove defects before the deployment of these applications since their faults may cause serious hazards. To this end, we have been studying testing techniques for deep learning technologies, and, in this paper, we propose a novel testing approach for Convolutional Neural Network (CNN) [1], one of the most widely used deep learning techniques for image recognition.

We claim that it is necessary to check the effect of every cell in a CNN before its deployment. One of the most straightforward criteria for this is **activation coverage** [2]; that is, every cell must be activated at least once by the images in test cases. Unfortunately, many cells in a CNN may remain uncovered in their training and verification partly because the number of samples in training and verification data is not enough and partly because there is a bias on sample data. Note that it is a very low probability that sample data provide enough edge cases.

To alleviate this phenomenon, we develop a test case generation approach for satisfying the following two conditions. The first condition is that the generated image test case should activate a target cell on CNN for checking its effect on the classification result of CNN. Of course, this condition is the most fundamental one for our purpose. Unfortunately, our experience showed that this condition is not enough for generating efficient test cases. Blindly generated images for just activating a target cell look like noisy patterns and do not resemble real images. Thus, even though these images activate the uncovered target cell, but it is difficult to decide the pass-fail of these images because we cannot assign expected values to these noisy images.

To mitigate this problem, we introduce the second condition to the test image generation. The second condition is that the generated image looks like a *real image* as a plausible test case. Of course, the resemblance of a real image is a subjective term, and, thus, it isn't straightforward to measure the reality of generated images. Fortunately, recent AI techniques provide remarkable capabilities for a fake image generation. So, by utilizing one of these AI techniques, it is possible to achieve the second condition.

Our approach is based on two research works, **activation maximization** [3] and **Generative Adversarial Network (GAN)** [4]. **Activation maximization** allows achieving the first condition by searching an input image that results in the maximum value of a target cell in a network. This technique is based on the gradient search used in deep learning, and, thus, we can easily implement it by leveraging the gradient search capability provided by deep learning frameworks. Current deep learning frameworks such as Tensorflow [11], PyTorch [12], and CNTK [13] provide the flexible implementation of the gradient search with high performance. In this paper, even though we use Tensorflow with the famous Python

E-mail:hhkim@sol-lin.com

frontend Keras [6], we strongly believe that the proposed technique can be implemented in other frameworks easily.

Then, we use **GAN** to achieve the second condition. GAN is one of the most famous fake image generation techniques. Our goal of the integration of GAN with activation maximization is to impose *plausibility constraints* on the gradient search. To our best knowledge, this is the first attempt to improve the quality of images using the generative modeling approach. For explaining the details of our approach, we use a CNN for the famous handwritten digits set MNIST [5].

The structure of this paper is as follows: First, Section 2 briefly summarizes activation maximization and GAN. Section 3 explains the definition of activation coverage and how to measure the coverage. Then, Section 4, the main part of this paper, describes the generation techniques for activation test cases step by step. First, we show the case of *activation maximization only* and discuss its limitation of plausibility. Secondly, we explain how to utilize the generator of GAN to overcome this limitation. Then, we present the integrated approach by defining multi-objective optimization for image test case generations for satisfying two conditions. Finally, Section 5 provides concluding remarks with some proposals for future researches.

2. BACKGROUND

In this section, we briefly explain two research works, *Activation Maximization* and *Generative Adversarial Network* (GAN), those form the cornerstones of the proposed approach.

A. Activation Maximization

The original goal of **activation maximization** is to generate a representative image of a class in an image classification neural network. For example, let us consider activation maximization on VGG16 [14], one of the most famous image classification networks. VGG16 can recognize 1,000 classes from various input images. When you put an image on VGG16, then, VGG16 predicts the class of your image by the probability array of the size 1,000. Higher the cell value in the array, the more likely the image is in the class of the cell. That is, the prediction of VGG16 takes an image and generates its probabilities of classes.

Activation maximization takes a reverse direction of the prediction. That is, activation maximization takes a class and generates an image that results in the maximum probability of the class. For example, class number 20 in VGG16 is 'water ouzel.' Figure 1. shows a generated image of class 20, water ouzel, by activation maximization in VGG16.

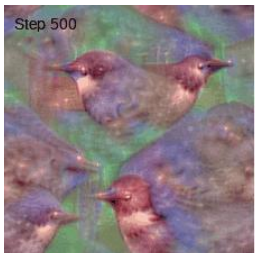


Figure 1. Activation Maximization of Water Ouzel (20) in VGG16[1].

The idea behind activation maximization is tricky but effective. Generally, when training a network *N*, we update the weights of *N* with gradients to achieve a minimum loss. There are a large number of loss functions for appropriately measuring the distance between the labels of inputs and their result of a network. Roughly speaking, we update *weights* with respect to inputs in training.

In activation maximization, we turn the direction of the training update. That is, we update *input* with respect to the weights of a classification neural network for maximizing the value of a classification cell. This formulation enables us to leverage the existing capabilities for gradient search and differential calculation in deep learning frameworks. Thus, we can implement this technique by a few modifications of existing functions and short lines of codes.

Note that, even though the image of Figure 1. shows representative parts of water ouzel and looks quite realistic, this is not a general case in test case image generation. In the case of Figure 1. , the target cell is the last layer of the network. So, the generated image contains every feature of the class corresponding to the target cell. However, when a target cell is not the last layer, then it captures only the weight of a partial combination of some features. In this case, the generated image looks unrealistic and artificial because the search of activation maximization only pursues the captured features by the target cell and does not consider other features.

B. Generative Adversarial Network

Generative Adversarial Network (GAN) [4] is one of the most famous generative modeling approaches that automatically discover and learn the latent distribution of sample images. After learning, GAN can generate a new plausible image that could have been drawn from the original sample images.

GAN consists of two competing networks referred to as **generator** and **discriminator** (Figure 2.). The role of the generator is to learn how to generate fake images that fool the discriminator. Meanwhile, that of the discriminator is to learn how to distinguish between fake and real images. As illustrated in Figure 2. , the generator takes a noise

[1] This image is generated by the example code from https://raghakot.github.io/keras-vis/

vector and generates a fake image while the discriminator takes an image and predicts whether it is real or fake.

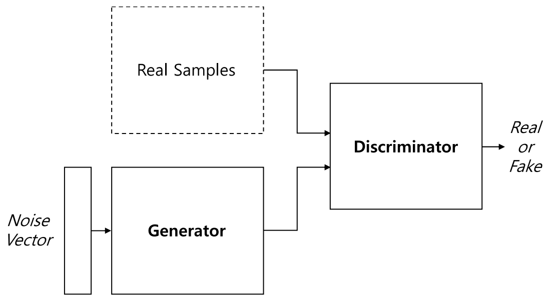


Figure 2. Generic Structure of Generative Adversarial Network

Generally, the training of GAN consists of the repetitions of the discriminator learning and generator learning. In the discriminator learning, we freeze the weights of the generator and train the discriminator on real and fake images with their labels. From this learning, the discriminator optimizes its weights for discrimination. Then, in the generator learning, we freeze the weights of the discriminator and train the generator to defraud the discriminator with generated images. To this end, we give a high penalty on the generated image when the discriminator predicts it as 'fake.' By updating the weights of the generator in the direction to reduce penalties, the generator improves its chance to defraud the discriminator.

As training progresses, the discriminator will no longer be able to identify the difference between fake and real images. Then, we can use the generator to create new realistic data that have never been observed before but follow the distribution learned from the sample images.

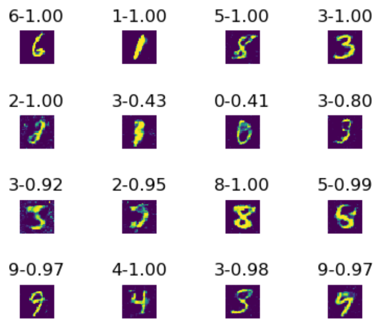


Figure 3. Fake Images in GAN for MNIST

Figure 3. shows the sample of fake images generated by the GAN. To evaluate the quality of fake images, we put generated images to our MNIST CNN and collect their classification results, digits in this case, and their prediction probabilities. We put the prediction result as a label for each generated image. That is, the label of the top left image in Figure 3 means that the classification of the image is digit 6, and the prediction probability is 100% (1.00). As illustrated in Figure 3. the quality of our GAN is quite good. That is, it is difficult to distinguish the fake images generated by our GAN from the real image in MNIST data.

3. Activation Coverage Testing for CNN

In this section, we explain how to define *activation coverage* with the MNIST CNN example. The idea of activation coverage is based on the fact that many layer types in deep neural networks have *activation functions* for introducing non-linearity on decisions. First, we explain the types of layers in MNIST CNN, and, then, we describe their activation functions.

TABLE I. The Structure of Exemplar MNIST CNN

Layer	Output	Activation Function	Activation Coverage
1. **Conv2d**	26, 26, 32	ReLu	Yes
2. **Conv2d**	24, 24, 64	ReLu	Yes
3. MaxPooling2d	12, 12, 64	-	No
4. Dropout	12, 12, 64	-	No
5. Flatten	9216	-	No
6. **Dense**	128	ReLu	Yes
7. Dropout	128	-	No
8. **Dense**	10	Softmax	Yes

TABLE I. shows the structure of MNIST CNN used in this paper. In this network, there are eight layers of five types. For each type, its description is as follows:

- **Conv2d**: This layer creates a 2D-convolution kernel. In short, a kernel is a matrix of weights. In this paper, we use the 3×3 kernel. The 2D convolution operation performs an elementwise multiplication of the kernel with the part of the 2D input and summing up the results into a single output value. Then, the layer applies an assigned activation function to the summed output value.
- **MaxPooling2d:** This layer performs a 2D pooling with the max filtering. In this paper, we use the 2×2 filter. For the part of the 2D input covered by the filter, this layer takes the maximum value of that part and generates a 2D output consisting of these maximum values. Because we use the 2×2 filter, the size of the 2D output is a quarter of that of the 2D input. This layer has no activation function.
- **Dropout**: This layer is a kind of regularization techniques for preventing over-fitting in training. In short, this layer randomly ignores the input in training. Generally, this layer does not work when predicting. This layer has no activation function.
- **Flatten**: This layer transforms the 2D input to the 1D output that has the same elements of the 2D input. This layer has no activation function.
- **Dense**: This layer performs a linear operation. The number of weights in this layer is the multiplication of

the output size with the input size. Also, this layer has a set of biases for each input. For example, the layer 6 in MNIST CNN has 1,179,648 (9,216×128) weights and 9,126 biases. For each output cell, this layer performs the multiplication of every input with weights and summing up the result with the bias into an output value. Then, the layer applies an assigned activation function to the output value.

Based on these characteristics of layer types, we exclude some layers from the measurement of activation coverage. Firstly, we exclude Layer 3 of **MaxPooling2D** because it has no activation function. The output of this layer can be derived from the input and has no non-linearity. Secondly, we do not measure Layers 4 and 7 of **Dropout** because they work in training only and has no activation function, again. Finally, we omit Layer 5 of **Flatten** because it has no activation function and only transforms the shape of the input. From this exclusion, we only measure Layers 1, 2, 6, and 8. In this case, the number of cells in the measurement is 58,634 (26×26×32 + 24×24×64 + 128 + 10). The types of the selected layers are **Conv2d** and **Dense**.

In these layers, there are two types of activation functions, **ReLu** and **softmax**. First, **ReLu** is the acronym for rectified linear unit and defined by the following function:

$$f(x) = \max(0, x)\ \textit{where } x \textit{ is input}$$

So, the graph of ReLu looks like Figure 4.

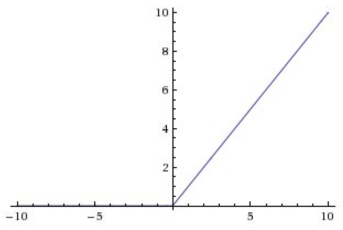


Figure 4. The Graph of ReLu Activation Function

Second, **softmax** is a function that transforms a vector of real numbers to a probability vector that sums to one. Before applying softmax, some values in the input vector could be negative or greater than one. Also, the input vector might not sum to 1. The function softmax transforms this input vector into an output vector that each component is in the interval (0,1), and the sum of all components in the output vector is one. So, the value of the output vector can be interpreted as probabilities.

Based on these characteristics of activation functions, we define the thresholds for activation coverage as follows:

- **ReLu**: Threshold is 0; that is, any output value greater than 0 is assumed to be activation.
- **softmax**: When the value of a cell *C* is a maximum in the vector, then the cell C is assumed to be activated; The maximum means that the cell *C* in the vector is most probable.

For measuring the activation coverage of Layers 1, 2, 6, and 8, it is necessary to modify the implementation of MNIST CNN to expose the output of target layers This modification is simple in our experiment environment Keras [6], and the famous book [7] written by the author of Keras provides a detailed explanation for it.

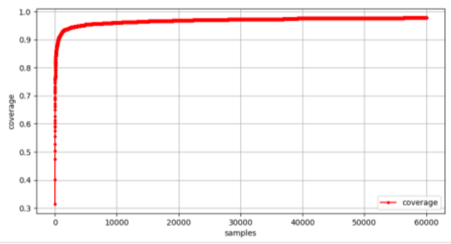


Figure 5. Activation Coverage of MNIST CNN for Training Data

Figure 5. shows the accumulated activation coverage of 60,000 training digit images in MNIST for MNIST CNN. Note that, as the number of tested images increases, the activation coverage of MNIST CNN grows quickly, but it is soon reaching its limit and not reaching 100% even though the size of the training set is huge. This result reveals that it is difficult to achieve 100% activation coverage.

4. IMAGE TEST CASE GENERATION FOR CNN

In this section, the main part of this paper, we explain the techniques for achieving the following two conditions by test image generation.

- The image should activate an uncovered cell in CNN for checking its effect.
- The image should resemble a real image for the pass-fail decision of testing.

First, we demonstrate the result of the case where we use activation maximization only. Even though this technique can satisfy the first condition, the resulting image looks absurd. Thus, we try to overcome this limitation by utilizing the generator of GAN in the next subsection. For some cells, this attempt generates an image that satisfies the first and second conditions at the same time. Finally, in the last subsection, we present the integrated approach for image test case generations.

A. Activation Maximization for Image Generation

For generating an image for activating a target cell, we can use the activation maximization by the gradient search. This image generation is conceptually simple, but there are some subtleties for implementation. First, we illustrate the core concept of the image generation, and, then, explain implementation subtleties.

As noted earlier, the activation maximization uses the gradient search capability of deep learning frameworks. For a given target cell *t* in CNN, we start with the blank image denoted by *image₀* as an input of CNN. For *image₀*, we get the output of *t* and calculate the gradient $g_0 = \frac{\partial\, t}{\partial\, image_0}$. Then, we can get the next *image₁* by $image_1 = image_0 + \gamma \cdot g_0$ with a learning rate γ. By generalizing this calculation, we can get the following image update formula:

$$image_{(i+1)} = image_i + \gamma \cdot g_i$$

We repeat the update formula until the value of *t* is larger than zero, that is, activated in **ReLu** activation function.

For implementing the activation maximation in Keras, there are two implementation subtleties. First, it is necessary to replace **ReLu** with the **linear** activation function. Note that the activation function **ReLu** outputs zeros for every negative input and, thus, its gradient for negative input is zero. As explained earlier, layers in neural networks perform the multiplication of input cells with weights and summing up the results with the bias into an input of their activation functions. Thus, the fact that a cell is not activated in a layer with **ReLu** activation function means that the input of its activation function is negative. In this case, the input image does not change because the gradient g_i is zero and, thus, $image_{(i+1)}$ is equal to $image_i$. That is, the gradient search gets stuck and does not progress. To make the search progress, we must replace **ReLu** with **linear** to get a non-zero gradient of a minus value.

Second, to leverage the optimizers provided by Keras, some modifications are necessary. Generally, in the training of neural networks, the input is constant, and the weights are variable for updating. However, in the activation maximization, the input must be variable while the weights of networks are constant. Because there is a strict distinction between variables and constants in the implementation of Keras, it is necessary to change the types of input and weights for using existing optimizers.

By applying this activation maximization technique, we can generate test images for activating uncovered cells by MNIST images. For simplicity of presentation, we focus on the test image generation for Layer 6 (Dense) of our MNIST CNN. As described in TABLE I. Layer 6 has 128 cells, and among these cells, 18 cells are uncovered by 60,000 images in MNIST training data. The following list enumerates the index of the uncovered cells in Layer 6.

- 1, 6, 14, 18, 19, 23, 47, 64, 67, 75, 98, 106, 108, 109, 112, 113, 120, 126

For understanding why MNIST test images do not activate these cells, it is helpful to examine the weights of these cells. Figure 6. illustrates the distribution of weights of Cell 126 in Layer 6. This cell has 9,216 weights for every cell in its previous layer. Because the number of negative weights is larger than that of positive weights, and because the input value of this cell is always positive, the operation of this cell results in negative output with a high probability. Thus, this negative output does not pass the threshold of **ReLu** activation function.

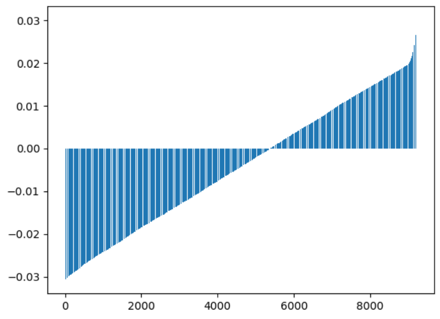


Figure 6. Weights of Cell 126 of Layer 6

It should be noted that it is theoretically difficult and infeasible in practice to decide whether the case to activate Cell 126 exists in real situations or not. When the case exists in real situations, but test data do not include it, we call it 'edge case' because it is rare. If the case does not exist in real situations, we call it 'impossible case.' Unfortunately, in many cases, the distribution of real situations is unknown and, thus, we cannot distinguish an edge case from an impossible case. So, we claim that it is necessary to test these uncovered cells for trying the detection of edge cases in the previous verification.

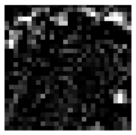


Figure 7. Generated Image for the Cell 126 of the Layer 6

Figure 7. is one of the generated images for Cell 126 of Layer 6. This image looks like just a spread of speckles. Note that Layer 6 is the last layer before the classification of digit images. It means that each cell of Layer 6 captures the significance of specific features in a digit image. Thus, when we try the activation of this cell only, the activation maximization technique does not consider the remaining features handled by other cells. In this case, the activation maximization technique results in a nonsensical image that signifies the selected specific features only.

This phenomenon occurs because the activation maximization technique in this subsection does not consider other features of ordinary digit images except those captured by a target cell. Figure 8. shows the current setting of the activation maximization technique, where there are no ways to incorporate additional features of digit image for inner layers. So, the resulting image has arbitrary intensities in its image plane that activate the features captured by the target cell only. When we apply activation maximization at the classification layer, located at the end of networks, this phenomenon is alleviated. Note that each cell of the classification layer is related to every feature of its corresponding digit. Thus, the resulting image contains a combination of features and may look similar to an ordinary digit.

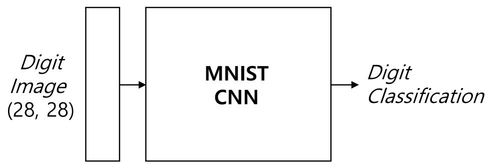


Figure 8. Activation Maximization Only Setting for MNIST CNN

To alleviate this phenomenon when tackling a target cell *t* in inner layers, we need a way to incorporate additional features for complementing the features captured in the target cell *t*. This need is the motivation of our second technique explained in the next subsection.

B. Activation Maximization with GAN

In this subsection, we try to overcome the limitation of the previous technique by providing a novel way to incorporate additional features of ordinary digit images.

The key idea of our approach is to use the *generator* of GAN for incorporating additional features of digit images. To our best knowledge, this GAN-based approach is a first attempt for test image generation in activation coverage.

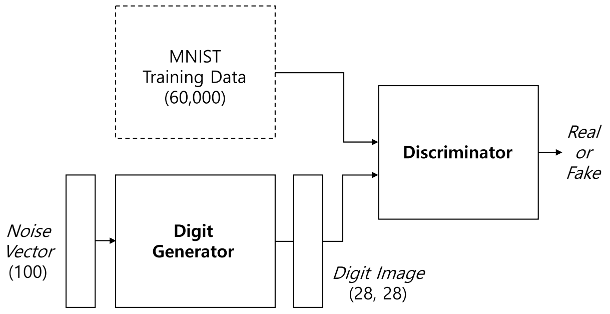


Figure 9. GAN for MNIST

First, for making a digit image generator, we train a GAN network with the structure in Figure 9. Like MNIST CNN training, we use 60,000 MNIST training data. The trained generator takes a noise vector with size 100 and synthesizes a 28×28 image according to the MNIST input specification. Figure 3. shows some samples of synthesized digit images by our digit generator in GAN.

For using the trained digit generator, we concatenate it with MNIST CNN, like Figure 10. In this setting, the result of the activation maximization technique is not a digital image of MNIST CNN but a noise vector of Digit Generator. In this setting, the backpropagation of gradient search in the activation maximization passes through Digit Generator. Note that Digit Generator learns how to mimic MNIST digit images in the GAN training, and the weights of Digit Generator capture this learned knowledge. Thus, this passage through the weights of Digit Generator incorporates additional features of digit images in the activation maximization technique.

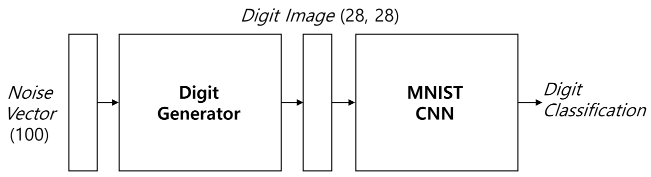


Figure 10. Activation Maximization with GAN Setting

The key benefit of the proposed approach is that there is no modification on neural networks and the algorithm. It is enough to connect trained two networks and possible to use the same algorithm for the activation maximization technique. Thus, it is easy to change the current digit generator with others trained by different machine learning techniques.

We apply the activation maximization technique in this setting with the modified version of *Adam* optimizer [15] in Keras. In this maximization, we use a learning rate 0.01 and maximum iteration 100,000. We choose Adam optimizer among various optimizers in Keras because this optimizer is the most advanced one and provides fast performance in most situations.

For the learning rate, we select this value through experiments. The learning rate controls the speed at which the model learns. For example, the symbol γ of our image update formula in the previous subsection is an example of learning rates. It is important to select an appropriate value for the learning rate. When a learning rate is too big, gradient descent search can inadvertently increase the training error. When a learning rate is too small, training is not only slower but also may become stuck at a valley of optimization. Roughly speaking, the learning rate 0.01 means that we update the original image with the 1% (1/100) of the gradient. Note that there are no theoretical ways to calculate the optimal learning rate currently, and it is a hot research topic. So, we perform some experiments for the gradient search in activation maximization and select 0.01 as a learning rate because this value shows a reasonable learning speed without sharp fluctuation.

Finally, we set a limit on the number of image update iterations in activation maximization. Again, in this case, we choose 10,000 based on our experience. In most cases, it is possible to get an image that covers the target cell before some thousand iterations, and the activation maximization terminates. However, sometimes, we cannot generate a test image even though we iterate a million times. Generally, the 10,000 iteration takes more or less 10 minutes in i5 CPU with 8 GB notebook, and, thus, we think this limit is reasonable for generating a test image.

In this setting, we perform test image generations and get activation images for 9 cells but fails for 9 cells among the uncovered 18 cells of Layer 6. Figure 11. shows the generated images with the label of the format '{index} {activation or not}.' For example, the label '1 True' means that this image is generated for Cell 1 of Layer 6 and succeeds in its activation. Even though test images look different from those of activation maximization only in the previous subsection, their quality is poor in comparison with synthesized images in Figure 3.

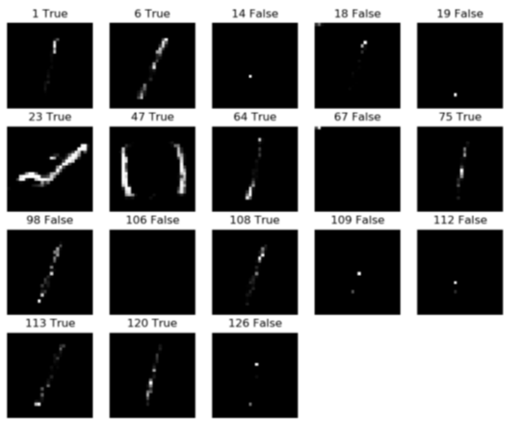
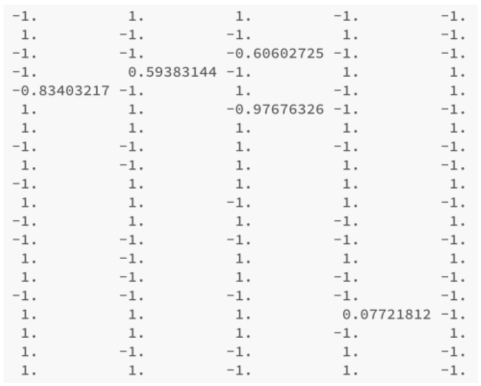


Figure 11. Activation Maximization with GAN Setting

The investigation of generated noise vectors shows why these vectors result in poor images. In GAN training, the values of a noise vector follow Uniform distribution with the range -1.0 ~ +1.0. However, in the activation maximization, the assumption of Uniform distribution on noise vectors does not hold. Figure 12. shows one of the generated noise vectors. As illustrated in this figure, most values among 100 cells in the noise vector are -1.0 or + 1.0, and only 5 elements are not edge values.

Further investigation shows that many values after gradient updates go over the limit of the range -1.0 ~ +1.0 but, for the validity of noise vectors, are clipped within the range. Because edge cases are necessary to cover inactivated target cells generally, these results, that is, many values in a noise vector are located at edges, look reasonable. However, to achieve the second condition, we must alleviate this phenomenon.



```
-1.          1.          1.         -1.         -1.
 1.         -1.         -1.          1.         -1.
-1.         -1.         -0.60602725 -1.         -1.
-1.          0.59383144 -1.          1.          1.
-0.83403217 -1.          1.         -1.          1.
 1.          1.         -0.97676326 -1.         -1.
 1.          1.          1.          1.          1.
-1.         -1.          1.         -1.         -1.
 1.         -1.          1.          1.         -1.
-1.          1.          1.          1.          1.
 1.          1.         -1.          1.         -1.
-1.          1.          1.         -1.          1.
-1.         -1.         -1.         -1.         -1.
 1.         -1.          1.          1.          1.
 1.         -1.          1.         -1.         -1.
-1.         -1.         -1.         -1.         -1.
 1.          1.          1.          0.07721812 -1.
 1.          1.          1.         -1.          1.
 1.         -1.         -1.          1.         -1.
 1.          1.         -1.          1.         -1.
```

Figure 12. Noise Vector Example of Activation Maximization

The most intuitive solution for this phenomenon is to make a noise vector follow Uniform distribution when activation maximization. However, we cannot find a practical way of how to impose this constraint on the gradient searching. Because the gradient search algorithm independently updates each value of a noise vector, it seems difficult to integrate Uniform distribution constraint with the gradient search algorithm efficiently. Instead of following Uniform distribution, we add another constraint for improving the image quality directly. The idea behind our approach is to use MNIST CNN as a quality checker of generated test images. Note that the final output of MNIST CNN denotes the probabilistic confidence of its prediction. So, we assume that higher confidence of test images, more realistic they are. In this paper, we use 0.7 as the confidence threshold for dropping poor quality images. Again, we choose this confidence threshold empirically.

Of course, this additional constraint lowers the probability that the generated image activates the target cell. Figure 13. shows the result of this additional constraint. As expected, this constraint improves the quality of images, but the rate of activation becomes lower than the original one. In this case, we can get only two activating images.

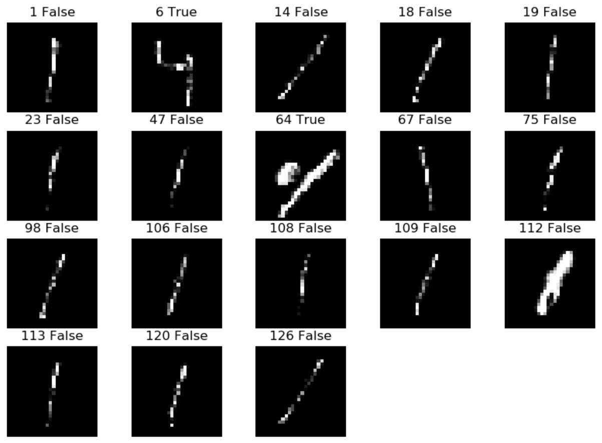


Figure 13. Activation Maximization with GAN and Confidence

TABLE II. CLASSIFICATION RESULTS WITH CONFIDENCE

Cells	Result	Confidence	Cells	Result	Confidence
1	1	0.92	75	1	0.72
6*	4	0.89	98	1	0.70
14	1	0.70	106	1	0.71
18	1	0.70	108	1	0.70
19	1	0.71	109	1	0.72
23	1	0.70	112	1	0.99
47	1	0.71	113	1	0.70
64*	4	0.72	120	1	0.70
67	1	0.71	126	1	0.71

TABLE II. summarizes the classification result of the generated images with their confidence. The activating images are marked with an asterisk in their cell numbers. The comparison of the images in Figure 11. with those in Figure 13. illuminates an interesting point on the characteristics of uncovered cells. For example, let us consider the case of Cell 1. Its image in Figure 11. succeeds in activation but that in Figure 13. fails. For convenience, Figure 14. shows the Cell 1 image of Figure 11. in the left with the label 'without confidence' and that of Figure 13. in the right with the label 'with confidence.'

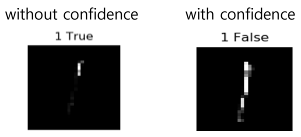


Figure 14. Image Test Case Generated by the Proposed Approach

The fact that the activating image in the left has fewer intensities than the failing image in the right implies that Cell 1 is related to a kind of *negative* features. That is, stronger intensities in pixels make down the probability of the activation of Cell 1. We can see this characteristic for every inactivated cell in Figure 13. This characteristic explains why these cells are hard to be activated in sample image data.

In this subsection, we develop the technique to achieve our second goal. To this end, we use the generator trained in GAN. To our best knowledge, the approach is a first attempt to utilize the generative modeling approach and provides a unique benefit that we can leverage the gradient search technique used in activation maximization. To improve the quality of test images, we use the target CNN as a quality checker. Even though this improvement does not impose the Uniform distribution constraint on GAN, it requires no additional efforts in training.

Unfortunately, the technique developed in this subsection does not provide satisfying activation coverage. For improving activation coverage as well as quality, we develop the integration of two techniques developed so far in the next subsection.

C. Image Test Case Generation Approach

In this subsection, we integrate the previous two techniques for satisfying two conditions of image test cases. For improving the activation coverage of generated test images and their quality, we develop a **two-phase approach**.

In the first phase, we use the activation maximization with GAN and confidence developed in the previous subsection. If we get an image that activates the target cell, then we stop the image generation. Otherwise, we perform the second phase for achieving activation coverage.

In the second phase, we return to the setting of activation maximization only but with different *input* and a different *loss* function in activation maximization. As an input, we use the generated image in the first phase. We claim that this input image is a good starting point because it is closer than the blank image to the activation of its target cell and looks similar to digit figures. So, this input results in a shorter time of the gradient search than the blank image.

As a loss function, we design a multi-objective optimization function for the following two goals. The first goal is that the generated image activates the target cell (**cell activation**), and the second goal is that the image is close to the input image (**figure resemblance**). To measure the distance of images, we use the sum of the *binary cross entropy* between pixels in images. We choose this distance measure because it is widely used in image comparison in machine learning fields. Then, we make a weighted sum of these two goals. We currently use 14/15 and 1/15 as the weights for the two goals, respectively. Again, we choose these weights empirically and these hyperparameters are easily adjustable.

Figure 14. shows a result of the proposed approach for 18 cells in Layer 6. The format of labels for generated test images is '{cell number} p1/p2-{activation}' where the keywords p1 and p2 mean its resulting phases. Let us examine this result. First, in this activation maximization, it is possible to get testing images for activating every uncovered cell. Second, among 18 images, phase 1 succeeds in only two Cells 1 and 64, and the others are the results of phase 2. Note that, in the previous subsection, we get two activating images, but their targets are Cell 6 and 64. That is, even though we use the same algorithm, but its results are different because of its non-deterministic nature. Thus, some repetition of the proposed approach is necessary to get satisfiable test images in practice. Thirdly, the visual quality of generated images quite varies. For example, testing images for Cell 14 or 18 lose the look of digit figures. As explained earlier, we use the multi-objective optimization for the two goals, that is, **cell activation** and **figure resemblance**. When the activation goal dominates the figure resemblance in the optimization of two objectives, the quality of images can be poor. In this case, we can mitigate this situation by adjusting the weights of these two goals in the second phase search. Of course, the optimization may take a longer time and fails to activate the target cell sometimes.

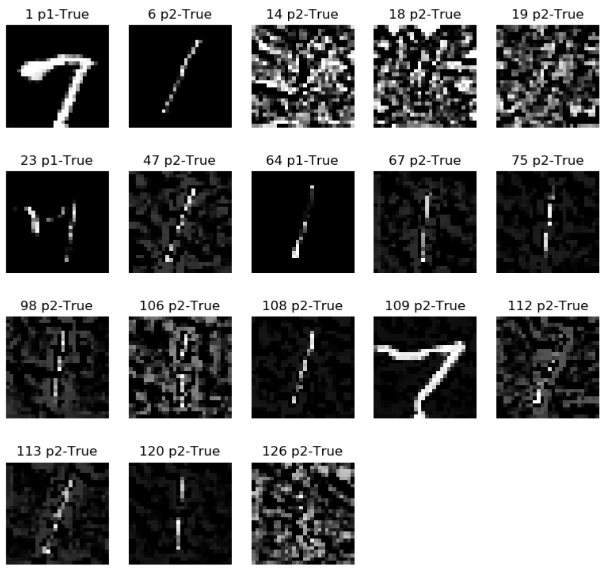


Figure 15. Image Test Case Generated by the Proposed Approach

Like the previous subsection, we provide the classification result of the generated image in TABLE III. Some numbers in this table are quite striking. For example, let us consider the images for Cells 14 or 18. We strongly believe that nobody may consider these images as digit figures. However, as shown in TABLE III. MNIST CNN provides high confidences for these images. Cells 14 and 18 have confidences 0.95 and 0.99, respectively.

Sometimes MNIST CNN may show unreasonable behavior for unexpected images. It is unfair to blame this behavior because we only provide plausible samples for its training. Thus, MNIST CNN doesn't know how to distinguish implausible images from real ones and does its best to predict the classification of implausible input images. However, it is risky to provide high confidence numbers for implausible images, and, thus, we feel that comprehensive testing for detecting these edge cases are significant before deployment.

TABLE III. CLASSIFICATION RESULTS WITH CONFIDENCE

Cells	Result	Confidence	Cells	Result	Confidence
1	7 (p1)	1.0	75	1	0.62
6	1	0.93	98	5	0.56
14	5	0.95	106	2	0.82
18	5	0.99	108	1	0.93
19	2	0.72	109	7	1.0
23	4(p1)	0.75	112	2	0.96
47	5	0.83	113	3	0.25
64	1	0.88	120	1	0.56
67	1	0.88	126	2	0.97

In this subsection, we develop the two-phase approach for trying to achieve two conditions. In the first phase of our approach, we use the activation maximization with GAN and confidence checking. If we do not get an activating image, then we use the activation maximization only with the multi-objective optimization. By this combination, we can get activating test images for every uncovered cell in Layer 6.

Even though the proposed approach improves the quality of generated test images, some generated images do not look like digits. This phenomenon raises roughly two questions. First is how to design the *plausibility checker* of generated test images. At least, this result indicates that our image quality checker, the confidence of MNIST CNN, has a weakness to pick out implausible images in some cases. If we provide an effective plausibility checker, it is possible to improve the quality of generated test images. The second question is a more fundamental one about the *decision* of whether an activating and plausible image exists or not. Because the input space of a CNN is generally extremely huge, it is impossible to test it exhaustively. So, if we have an effective tool for this decision, we can early stop the generation process of testing images for that target and save our effort.

5. Concluding Remarks

In this paper, we presented a test image generation approach for CNNs. The goal of our approach has to generate an image satisfying the following two conditions. First, this image should activate a specific target cell on CNN for checking the effect of this cell. Second, this image should look like a real image as a plausible test case. Note that it is difficult to achieve 100% activation coverage with a just given training or testing data and, thus, any image generation techniques of this type are crucial for locating locate defects for CNN.

Our approach is based on two techniques, activation maximization and Generative Adversarial Network (GAN). Activation maximization technique allows achieving the first condition while GAN supports the pursuit of the second condition by imposing plausibility constraints on image generation. To our best knowledge, our approach is a first attempt to utilize the generative modeling approach. Based on these two techniques, we develop a two-phase approach for trying to achieve the two conditions. To this end, we introduce the quality checker using confidence and multi-objective loss function.

There are some future research directions for improving our approach: First, it looks promising for incorporating more advanced styles of GAN, such as CycleGAN [8] or Style-based GAN [9] for fine controlling the constituents of images. Because these advanced GANs provide a so-called disentangled noise vector, we expect that they support more effective searching and more excellent results. Second, it is possible to use other generative techniques in the role of GAN. For example, Variational Auto-Encoder (VAE) [10], another famous generative model, can be used for imposing the figure constraints on image generation. Third, it is necessary to define the notion of plausibility in test images and to design the plausibility checker of generated test images. Lastly, even though it is very challenging, we hope that we get a practical way to decide whether an activating and plausible image exists or not.

Acknowledgment: This work was supported by the Ministry of Trade, Industry and Energy of the Republic of Korea
(Project 20006869 - Development of Self Driving Parts and Vehicle Mounting Technology for Large Bus).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks", In NIPS 2012.

[2] Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems", SOSP'17, 2017.

[3] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network". Dept. IRO, Université de Montréal, Tech. Rep, 4323, 2009.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. WardeFarley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks", In NIPS, 2014.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner "Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278-2324. 1998.

[6] C. Francois and others, Keras, https://keras.io.

[7] C. Francois, Deep Learing with Python, 1st Edition, Manning Publications, 2017

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efro, "Unpaired image-to-image translation using cycleconsistent adversarial networks", arXiv preprint arXiv:1703.10593, 2017.

[9] T. Karras, S. Laine, and T. Aila, "A Style-Based Geneator Architecture for Generative Adversarial Network", in CVPR 2019.

[10] D. P. Kingma and M.Welling, "Auto-encoding variational bayes", arXiv preprint arXiv:1312.6114, 2013

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: Large-scale machine learning on heterogeneous systems", 2015. Software available from tensorflow.org.

[12] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam, "Automatic differentiation in PyTorch', 2017

[13] Amit Agarwal, Eldar Akchurin, Chris Basoglu, Guoguo Chen, Scott Cyphers, Jasha Droppo, Adam Eversole, Brian Guenter, Mark Hillebrand, T. Ryan Hoens, Xuedong Huang, Zhiheng Huang, Vladimir Ivanov, Alexey Kamenev, Philipp Kranen, Oleksii Kuchaiev, Wolfgang Manousek, Avner May, Bhaskar Mitra, Olivier Nano, Gaizka Navarro, Alexey Orlov, Hari Parthasarathi, Baolin Peng, Marko Radmilac, Alexey Reznichenko, Frank Seide, Michael L. Seltzer, Malcolm Slaney, Andreas Stolcke, Huaming Wang, Yongqiang Wang, Kaisheng Yao, Dong Yu, Yu Zhang, Geoffrey Zweig (in alphabetical order), "An Introduction to Computational Networks and the Computational Network Toolkit", Microsoft Technical Report MSR-TR-2014--112, 2014.

[14] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", International Conference on Learning Representations, 2015

[15] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization", the 3rd International Conference for Learning Representations, San Diego, 2015.

Hyung Ho Kim received the B.S. degree from Sogang University and M.S and Ph.D. from KAIST in computer science. He is a vice president and principal consultant at Solutionlink, a system and software engineering consulting company in South Korea. After cofounding this company in 2000, he has been working on the development of safety critical system and software for various ICT and automotive companies including Hyundai Motors, LG, and Samsung, Mobis, and Mando. He is currently interested in safe architectural design on automotive system.