An Adaptive Threshold for the Canny Algorithm with Deep Reinforcement Learning

KEONG-HUN CHOI1 and JONG-EUN HA2
1Graduate School of Automotive Engineering, Seoul National University of Science and Technology, Seoul, Korea
2Department of Mechanical and Automotive Engineering, Seoul National University of Science and Technology, Seoul, Korea

Corresponding author: Jong-Eun Ha (e-mail: jeha@seoultech.ac.kr).

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government. (MSIT) (2020R1A2C1013335)

ABSTRACT The Canny algorithm is widely used for edge detection. It requires the adjustment of parameters to obtain a high-quality edge image. Several methods can select them automatically, but they cannot cover the diverse variations on an image. In the Canny algorithm, we need to set values of three parameters. One is related to smoothing window size, and the other two are the low and high threshold. In this paper, we assume that the smoothing window size is fixed to a predefined size. This paper proposes a method to provide adaptive thresholds for the Canny algorithm, which operates well on images acquired under various variations. We select optimal values of two thresholds adaptively using an algorithm based on the Deep Q-Network (DQN). We introduce a state model, a policy model, and a reward model to formulate the given problem in deep reinforcement learning. The proposed method has the advantage that it can adapt to a new environment using only images without labels, unlike the existing supervised way. We show the feasibility of the proposed algorithm by diverse experimental results.

INDEX TERMS Edge detection, deep Q-network, deep reinforcement learning.

I. INTRODUCTION Various vision tasks such as image segmentation, object detection, and recognition require good edge detection. The Canny algorithm is widely used for edge detection [1]. It requires setting three parameters related to the window size of smoothing and two thresholds related to the hysteresis process. Figure 1 shows various edge images by different values of two thresholds of the Canny algorithm. We can notice dramatic differences in resulting edge images according to the variation of the two thresholds. Therefore, it is necessary to select suitable values of three parameters to obtain the best edge image.

In this paper, we propose a method that gives adaptive thresholds appropriate for an image. Among three parameters in the Canny algorithm, we use a fixed value for the parameter related to the smoothing window size. We adopt the DQN [2, 3] and propose the state and policy model necessary for applying the DQN to the given problem. Also, we present the reward model consisting of a convolutional neural network (CNN). Supervised learning-based methods also can be considered to solve this problem. But, supervised learning requires ground truth labels. We could consider a CNN, which gives values of two thresholds using an image as input. In this case, it involves ground truth labels of values of two thresholds per image. It requires a lot of time to prepare them for thousands of images. Besides, supervised learning shows a good performance only in environments similar to training ones. One way to adapt to a new environment is to train again using images from the new environment. On the contrary, the proposed algorithm can adapt to new environments through additional training using only images with unsupervised learning. To the best of our knowledge, this is the first attempt to apply deep reinforcement learning to the automatic selection of values of parameters in the Canny algorithm.

The proposed algorithm can provide suitable thresholds values in the Canny algorithm for images acquired from diverse environments. We can use the resulting edge images for high-level object detection, image segmentation, 3D reconstruction, and motion analysis [41, 42, 43, 44]. For example, edge detection from aerial images and remote sensing is used for inferring the urban metrics, which are the basis of urban planning strategies [43, 44].
The contributions of the proposed algorithm can be summarized as follows. (1) A deep reinforcement learning method is proposed to select appropriate values of parameters in the Canny algorithm automatically. (2) We propose the state model, the policy model, and the reward model necessary to apply the DQN to the given problem. (3) The proposed algorithm can adapt to new environments different from training environments in an unsupervised learning.

![Original image and resulting edge images according to the variation of values of two thresholds in the Canny algorithm](image)

**FIGURE 1.** Original image and resulting edge images according to the variation of values of two thresholds in the Canny algorithm (a) original image (b) H:60, L:30 (c) H:138, L:69 (d) H:220, L:200.

### II. RELATED RESEARCH

We can categorize most edge detection algorithms into filter-based, learning-based, and recent deep learning-based algorithms.

Traditional filter-based algorithms [1, 4] find edges by checking sudden changes in intensity, color, texture, etc. Learning-based algorithms detect edges by using supervised models and hand-crafted features. Statistical Edges [5], Pb [4], and gPb [6] use information theory on top of features derived by careful manual design. Early learning-based methods such as BEL [7], Multi-scale [8], Sketch Tokens [9], and Structured Edges also heavily rely on features designed manually. Dollar et al. [10] detect structured edge that jointly learns ground truth clustering and the mapping of image patch to a clustered token. Their algorithm showed state-of-the-art performance on the BSDS500 dataset before the advancement of deep learning-based algorithms.

Recent deep learning-based algorithms utilize features that are generated by convolutional neural networks (CNN) [11]. Bertasius et al. [12] use CNN to find features of candidate contour points. Xie et al. [13] propose holistically-nested edge detection (HED) that integrates the outputs from different intermediate layers with skip connections. Xu et al. [14] use a hierarchical model to find multi-scale features fused by a gated conditional random field. He et al. [15] propose a Bi-Directional Cascade Network (BDCN) structure to detect edges at different scales. They train the network using other labeled edges for each scale. They introduced Scale Enhancement Module (SEM), which utilizes dilated convolution to generate multi-scale features instead of using deeper CNNs or explicitly fusing multi-scale edge maps. Recent algorithms for edge detection focus on the accurate detection of object boundaries that can provide semantics cues for further processing, such as object detection, segmentation, and tracking. This paper deals with the automatic selection of two Canny algorithm parameters that guarantee stable edges on an image under significant variations caused by illumination and pose variation.

Lu et al. [16] proposed an algorithm to select thresholds for the Canny algorithm using a histogram of the gradient image. Fang et al. [17] proposed an algorithm to choose a high threshold for the Canny algorithm using Otsu method [18]. They cannot select a low threshold. Huo et al. [19] proposed an algorithm to determine a high and low threshold for the Canny algorithm. They choose a low threshold using a probability model. Lu et al. [20] adaptively select two thresholds using minimal meaningful gradient and maximal meaningless gradient magnitude assumption.

Yitzhaky and Peli [35] proposed an algorithm to select the best edge parameters. First, they construct Estimated Ground Truth (EGT) with different detection results. Then, they determine the optimal parameter set using a Chi-square test. Medina-Carnicer et al. [36] proposed an algorithm for the unsupervised determination of hysteresis thresholds by fusing the advantages and disadvantages of two thresholding algorithms. They find the best hysteresis thresholds in a set of candidates. Medina-Carnicer et al. [37] proposed a method to determine hysteresis thresholds of the Canny algorithm automatically. Therefore, it can be used as an unsupervised edge detector.

These traditional unsupervised methods have the advantage that they can be applied without a learning process. However, in the case of these methods, only evaluation results for a small number of images are provided. It is necessary to evaluate them using a lot of data, as in deep learning, to evaluate these algorithms’ performance objectively.

Reinforcement learning [21] shows a good performance in temporal decision-making problems. In a typical reinforcement learning, an agent aims to learn a policy that gives maximum accumulated reward from an environment. Recently, the integration of reinforcement learning and deep learning showed human-level control [22]. Deep Q-Networks (DQN) [2, 3] showed human-level control is possible on Atari games. Deep reinforcement learning offers impressive successes on various tasks such as playing the board game GO.
III. PROPOSED METHOD

We regard an image as a state, and we consider that the agent has two actions corresponding to a high and low threshold in the Canny algorithm. Overall, we present two different methods according to the definition of the state and action model. The first method is related to the state model that considers an image and two thresholds together. We call it the multi-step method because values of two thresholds are searched multiple times for a given image. The second method is related to the state model that considers an image alone as a state. We call it the single-step method because values of two thresholds are searched only once for a given image. We introduce an action model according to the definition of the state model.

A. THE STATE MODEL

In the initial DQN [2], the goal was to generate control values for a game, and they used four consecutive images in the game display as states. When the agent interacts with the environment, they consider a changed game screen as the state at the next time step. In the game case, we can regard the game screen before and after the action as the current and next state, respectively. If we follow this approach, we could consider the input image as the current state and the resulting edge image as the next state. This state definition may cause problems during training because the original image used as an input and the edge image resulting from the action belong to different domains.

We propose two types of state definitions to solve this problem. Table 1 shows two definitions of the current state and the next state. In Table 1, the multi-step method refers to updating the parameters of a network several times using the same image. The single-step method refers to a form that corrects the parameters of a network only once using the same image. When we consider the input image and the values of current thresholds as the state, only the values of thresholds are updated. Among two elements in the state, the values of two thresholds change while the image remains fixed to fit them into the existing DQN structure without significant change. In the case of a single step, both the current and next state consists of an image. Since we do not include thresholds in the state, we consider an image randomly selected as the next state. This state definition may cause problems during training caused by too many actions.

In the case of two thresholds in the Canny algorithm, one is related to the high threshold value, and the other is related to the low threshold value. It must be satisfied that the low threshold value is smaller than that of the high threshold. In the dependent type, we reflect this constraint by selecting the low threshold value according to the high threshold value. We set the range of the low threshold value from 0 to the value of the high threshold.

| Table 1. The definition of the state in the multi-step and single-step method. |
|-----------------|-----------------|
| Method          | Current state   | Next state     |
| Multi-step      | Image and two thresholds | Image and two thresholds |
| Single-step     | Image           | Image          |

B. THE ACTION MODEL

We define two types of actions according to the state's definition in Table 1. Table 2 shows the configuration of action that matches the state model of the multi-step method. It consists of five different types. An action corresponding to one threshold can take one among three increasing thresholds, decreasing thresholds, and no value change. As a result, we design the multi-step method to have five action types by combining two thresholds, as shown in Table 2.

In the case of a single-step method, action means a specific combination of values of two thresholds. Table 3 shows the two action types used in a single-step approach. We classify two types as independent and dependent types. In the independent type, the high threshold value affects the low threshold value in the Canny algorithm. In the independent type, values of high and low thresholds do not affect each other. The range of values of each threshold in the Canny algorithm belongs to an integer between 0 and 255. Therefore, the number of combinations of values by two thresholds amounts to 65,536. If we use all combinations by values of two thresholds as discrete action, difficulties can occur during training caused by too many actions.

In the case of two thresholds in the Canny algorithm, one is related to the high threshold value, and the other is related to the low threshold value. It must be satisfied that the low threshold value is smaller than that of the high threshold. In the dependent type, we reflect this constraint by selecting the low threshold value according to the high threshold value. We set the range of the low threshold value from 0 to the value of the high threshold.

| Table 2. The definition of action in the multi-step method. |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| Type 1          | Type 2          | Type 3          | Type 4          | Type 5          |
| High threshold  | +1              | -1              | 0               | 0               | 0               |
| Low threshold   | 0                | 0               | +1              | -1              | 0               |

| Table 3. The definition of action in the single-step method. |
|-----------------|-----------------|-----------------|
| Type            | Number of action for high threshold | Number of action for low threshold |
| Dependent       | 256             | 20              |
| Independent     | 256             | 256             |

C. CONFIGURATION OF POLICY NETWORK

In deep reinforcement learning, actions are selected by referencing the policy network. Q(s, a) denotes Q value achievable by action taken at the current state. Usually, an action that gives the largest Q value is selected. We propose two types of policy networks according to the state and action definition in Tables 1, 2, and 3.

Figure 2 shows a policy network used in the multi-step method. The input of the network is an image and two thresholds. We process an image through the network while the two thresholds do not go through the network. Two thresholds are merged in the network through concatenation before the final stage, as shown in Figure 2. The output of the network is Q values corresponding to five action types in Table 2. Figure 3 illustrates a policy network used in the single-step method.

In the Canny algorithm, the number of combinations by two thresholds amounts to 65,536. When we use all combinations for outputs of the policy network, problems may occur during training. To solve this problem, we propose the network that yields the Q value of each threshold independently. The proposed policy network can treat two different action types in Table 3 using the same network.
structure. We use the VGG19 [38] to extract feature vectors from the input image in the earlier network and add fully connected layers after the VGG19 layers. We apply global max pooling between the VGG19 and the fully connected layer. During the training of the policy model, we fix parameters related to the VGG19 and only update parameters associated with the remaining layers.

In the Canny algorithm, the low threshold value is smaller than that of the high threshold. We reflect this constraint in the proposed policy network, as shown in Figure 3. In the dependent type of Table 3, the number of actions related to the high and low threshold is 256 and 20, respectively. Therefore, we could have 5,120 Q values from the combination of the output of the network. We select actions that give the maximum Q value among these combinations. In the independent type of Table 3, the number of actions related to the high and low threshold is 256 and 256, respectively. Therefore, we could have 65,536 Q values from the combination of the output of the network. Among these combinations, we choose actions that give the maximum Q value.

D. REWARD COMPUTATION

In deep reinforcement learning, an agent's reward after an action in a given state is necessary for training. To compute the reward, we use the CNN from our previous work [30]. It produces a value between 0 and 1 using edge image as input. We denote the output of the network as edge quality. Figure 4 shows the CNN used to compute reward, consisting of the ResNet50 [31] and fully connected layers. In our previous work [30], we trained the model using a total of 4,020 images where it consists of 2,120 good edge images and 1,900 bad edge images. In this paper, we train the model using more training images to improve accuracy.

We use images from the BDD100K [32] for training. The procedure for the preparation of training images is as follows. We obtain multiple edge images per image using the Canny algorithm by varying values of parameters. Then, we manually select good and bad edge images for training. Specifically, we divide the range of threshold by increment 4. We obtain 4,096 combinations of values of two thresholds. Finally, we accept 2,080 cases by removing cases where the low threshold value is larger than the high threshold. Finally, we manually select the samples of good and bad edge images. Throughout this process, we prepared 16,000 images for training. We do the training up to 20 epochs, and we stop training when the accuracy of validation decreased more than two times. Figure 5 shows original images, output values by the network, and corresponding values of two thresholds used in the Canny algorithm.

The output of the network in Figure 4 has a value between 0 and 1. Directly using this value as a reward can cause problems in deep reinforcement learning. Since all reward values are positive, training cannot correctly proceed for the behavior that generated negative results. In addition, in the case of independent type in the single-step method of Table 3, an issue in which the high threshold value is lower than that of the low threshold may occur. A strategy to cope with this problem is necessary. Table 4 shows the configuration of reward values to solve this problem. We design the final reward value to have positive and negative ones. When the high threshold value is larger than the low threshold, the reward value takes one among three values according to the network's output, as shown in Table 4. In the opposite case, the reward value is set as -1 to suppress those types of actions.
FIGURE 3. The proposed policy network for the single-step method.

FIGURE 4. A CNN model [30] for the evaluation of edge quality.

FIGURE 5. The original image, the high and low threshold value used in the Canny algorithm, and output value by the network by Figure 4.

| Original Best Worst 1 Worst 2 |
|-----------------------------|-----------------|-----------------|-----------------|
| H:255, L: 208 edge quality: 1.00 | H: 182, L: 0 edge quality: 0.09 | H: 8, L: 0 edge quality: 0.00 |
| H: 89, L: 56 edge quality: 1.00 | H: 230, L: 228 edge quality: 0.17 | H: 8, L: 0 edge quality: 0.00 |
| H: 93, L:24 edge quality: 0.82 | H: 194, L: 172 edge quality: 0.34 | H: 2, L: 0 edge quality: 0.00 |

TABLE 4. The configuration of reward value.

| Action   | Network output | Reward |
|----------|----------------|--------|
| High ≥ Low | ≥ 0.99         | 1.0    |
|          | ≥ 0.5          | 0.1    |

IV. EXPERIMENTAL RESULTS

We resize all images to a size of 256(H)X256(W)X3(C) for training, and we convert values on the image into the range from 0 and 1. Computation was done using a computer with NVIDIA 2080TI and Intel i7-6700k. We regarded one episode as the case when we used all images for the training. We add the procedure to cope with the same action repeated multiple times during training in the multi-step method. We monitor previous action behaviors during training, and we give high rewards for stopping the action when the same action is repeated three or more times. We use Adam [33] as the optimizer in training.

A. RESULT OF A MULTI-STEP METHOD

We train the presented model using 70,000 images from BDD100K [32]. We evaluate experimental results by checking two values. One is the change of reward value on the same image during training, and the other is the average reward value at the final step of each episode. In the multi-step method, we regard the original image and two thresholds as the state, and the action configuration is shown in Table 2.

Table 5 shows the results of the multi-step method according to different initial values of two thresholds. For each case, we offer the mean reward value at the final step and computation time. When the initial values of the two thresholds were 128 and 128, it took 2.7 seconds for the model to produce output. In this case, the network requires...
about 77 steps for each image, and the average time needed for each step is 40ms.

Figure 6 shows the resulting edge image according to different initial values of two thresholds. We offer selected values of two thresholds for each image. Figure 7 shows the variation of the last 30 reward sum of using image #24, image #491, and image #584 along step according to episode. Figures 7(a) and 7(b) show the variation in episode 1 and episode 37, respectively. We notice that reward increases incrementally along a step at the latter stage while remaining minus at the earlier stage during training. All three images show a similar tendency. Figure 8 shows the change of reward value using image #491 at the final step of each episode. We can notice that the reward value increases as the training progresses, and through this, we can conclude that the proposed multi-step method can solve the given problem. From these results, we can conclude that we can use the proposed multi-step method to select values of two thresholds in the Canny algorithm automatically. Still, it has a shortcoming that it takes a lot of time to process a single image.

**Table 5. Result of a multi-step method according to different initial values of two thresholds.**

| Initial values of two thresholds (H, L) | Mean reward at last step | Computation time per image [sec] |
|----------------------------------------|--------------------------|---------------------------------|
| (196, 196)                             | 0.74                     | 2.51                            |
| (128, 128)                             | 0.77                     | 2.68                            |
| (64, 64)                               | 0.64                     | 3.02                            |

**B. RESULT OF SINGLE-STEP METHOD**

The proposed single-step method uses images as the current state and the next state and uses actions shown in Table 3. We use two types of actions in Table 3 using the same network shown in Figure 3. We investigate the reward variation for all steps rather than the variation of reward per episode since we randomly choose images corresponding to the current state and next state. The average computation time of 3,000 images five times takes 118.341 seconds for the [256, 20] action model and 117.291 seconds for the [256, 256] action model, respectively. The single-step method takes 39ms, while the
multi-step method takes 2,700ms for processing one image. Therefore, the single-step method is more efficient than the multi-step method in computation.

Figure 9 shows the change of mean reward of 50 images along step by [256, 20] action model and [256, 256] action model during training, respectively. Table 6 shows the comparison results of the proposed methods and the supervised method [30]. We obtain various statistics by evaluating 5,120 images. In the case of edge quality, the supervised learning method [30], the [256, 20] action model, and the [256, 256] action model give 0.569, 0.812, and 0.840, respectively. The single-step method using the [256, 256] action model provides the best result. Figure 10 shows the edge images generated by each method. Figure 10 shows values of low and high thresholds generated by the proposed single-step method at the bottom of an image. The ground truth edge image is manually selected. In supervised learning, we can observe that most of the edges on complex shapes are lost. On the other hand, we can notice that the proposed method extracts edges well, even in harsh conditions.

![Figure 9](image1.png)

**FIGURE 9.** The variation of mean reward of 50 images (a) [256, 20] action model (b) [256, 256] action model.

![Table 6](image2.png)

**TABLE 6.** The comparison result of the single-step method to the supervised method.

| Algorithm            | Precision | Recall | FM   | Edge quality |
|----------------------|-----------|--------|------|--------------|
| [256, 20] action model | 0.694     | 0.986  | 0.814| 0.812        |
| [256, 256] action model | 0.737     | 0.971  | 0.838| 0.840        |
| Supervised [30]      | 0.912     | 0.683  | 0.781| 0.569        |

![Figure 10](image3.png)

**FIGURE 10.** Comparison of edge image by various algorithms (a) original image (b) supervised learning [30] (c) [256, 256] action model (d) [256, 20] action model (e) ground-truth.

C. EVALUATION OF GENERALIZATION POWER

We evaluate the generalization power by applying a trained model to images acquired from environments different from training ones. We use the BDD100K [32] dataset for training and the CDnet2014 [34] dataset for testing to evaluate the generalization power. The CDnet2014 dataset is used in visual surveillance, while the BDD100K dataset is used in autonomous navigation.

Table 7 shows the comparison results by each algorithm. We evaluate by comparing the edge quality, which the CNN in Figure 4 computes. We use 11,150 images from the four scenes of the ‘Highway’, ‘Office’, ‘Wet Snow’, and ‘Skating’ scenes in the CDnet2014 dataset for evaluation. In the supervised method [30], we notice that the performance significantly degrades in an environment different from training. On the other hand, we can see that the proposed method shows a slight drop in performance in an environment different from the training one. Additional training is required.
to improve the performance using images acquired from the new environment under the supervised method. It involves ground truth labels corresponding to input images. On the other hand, the proposed method can train using only images acquired from a new environment without ground truth labels.

Table 7 shows the performance of the proposed method with and without additional training in a new environment. We use 80% of the total image to 8,920 images for further training by the proposed method. The remaining 20% amounts to 2,230 images are used for evaluation. In additional training in the new environment, we fix the parameters related to VGG19 layers and only update the remaining parameters. We could obtain 5.3% improvement through additional training in a new environment. Table 8 shows the variation of edge quality according to the epoch in additional training. There is an improvement in the earlier stage, but further improvement does not happen as training continues. We think that retraining the model in Figure 4 in new environments would give further improvement.

Figure 11 shows the edge images produced by each algorithm using images acquired from environments different from training. Figure 12 shows comparison results of edge images with and without retraining in a new environment. The value noted at the bottom of the image corresponds to the edge quality. We can notice that we can have clear improvement by retraining in a new environment.

**Table 7.** Evaluation result using images not used in training.

| Model                        | Edge quality |
|------------------------------|--------------|
| Supervised [30]              | 0.45         |
| [256, 256] action model without retraining | 0.72 |
| [256, 256] action model with retraining     | 0.76 |

**Table 8.** The variation of edge quality according to an epoch in additional training.

| Train epoch | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |
|-------------|----|----|----|----|----|----|----|----|----|
| Edge quality| 0.705 | 0.721 | 0.743 | 0.752 | 0.763 | 0.762 | 0.764 | 0.761 | 0.759 |

**D. EVALUATION USING GROUND TRUTH CANNY EDGE IMAGES**

We compare to other algorithms using Heath’s set of images [38] where ground truth edge images by the Canny algorithm are determined manually. Baddeley’s measure [39] is used to quantify the performance of each method. It is defined as follows.

\[
D_w^p(A, B) = \left( \frac{1}{N} \sum_{x \in X} w(d(x, A)) - w(d(x, B)) \right)^{1/p}
\]  

(1)
$d(x, A)$ denotes the shortest distance from $x \in X$ to $A \subseteq X$. $w(t)$ is a continuous function with concave and strictly increasing properties. $D_w^c(A, B)$ can have a value from 0 to 1. For similar images $A$ and $B$, $D_w^c(A, B)$ gives a value close to 0. Baddeley [39] used the transformation $w(t) = \min\{t, c\}$ for a fixed $c > 0$. We denote $H$ and $W$ as the width and height of an image. The values $c = \sqrt{H^2 + W^2}$ and $p = 2$ were used in the evaluation.

Table 9 shows comparison results of the proposed algorithm to three other algorithms using Baddeley’s measure. SOHT denotes subset and overset for hysteresis thresholds [36]. UTHT denotes unimodal thresholding for hysteresis thresholds [40]. Evaluation metrics of SOHT, UTHT and [37] in Table 9 are those specified in [37]. The proposed algorithm uses images of $256(H)\times256(W)\times3(C)$ size as the input of a network. Heath’s set of images are provided as gray images. We experimented with the proposed algorithm in two ways. First, we reduced the original gray images into color images to approve their power.

In Table 9, bold black and red correspond to the best and worst case for each image. The traditional unsupervised method [37] shows the best performance. It has the advantage that it does not require training. The proposed algorithm is based on learning using images obtained in various conditions. Therefore it can operate on them as deep learning algorithms in image classification, and object detection has shown. Unsupervised algorithms and the proposed algorithm need to be tested using more images as other fields in deep learning to approve their power.

Fig. 13 shows the ground truth edge images and results by the proposed algorithm for some representative images. For orange and beehive images, the proposed algorithm gives worse outcomes than other algorithms. For trashcan and stairs images, the proposed algorithm gives better results than different algorithms.

| Image       | SOHT [36] | UTHT [40] | Proposed (gray→color) | Proposed (color→color) |
|-------------|-----------|-----------|------------------------|------------------------|
| Golf-cart   | 0.0248    | 0.0366    | 0.0300                 | 0.0153                 |
| Pitcher     | 0.0095    | 0.0070    | 0.0071                 | 0.0087                 |
| Stapler     | 0.0026    | 0.0083    | 0.0019                 | 0.0121                 |
| Mailbox     | 0.0450    | 0.0336    | 0.0339                 | 0.0042                 |
| Pillow      | 0.0178    | 0.0194    | 0.0195                 | 0.0143                 |
| Brush       | 0.0280    | 0.0300    | 0.0133                 | 0.0222                 |
| Grocery Cart| 0.0160    | 0.0223    | 0.0177                 | 0.0157                 |
| Tire        | 0.0170    | 0.0286    | 0.0267                 | 0.0086                 |
| Grater      | 0.0274    | 0.0175    | 0.0176                 | 0.0159                 |
| Basket      | 0.0046    | 0.0049    | 0.0031                 | 0.0080                 |
| Orange      | 0.0157    | 0.0080    | 0.0086                 | 0.1228                 |
| Banana      | 0.0120    | 0.0154    | 0.0137                 | 0.0204                 |
| Egg         | 0.0106    | 0.0116    | 0.0114                 | 0.0456                 |
| Elephant    | 0.0026    | 0.0024    | 0.0030                 | 0.0312                 |
| Pend        | 0.0217    | 0.0376    | 0.0260                 | 0.0105                 |
| Firecracker | 0.0229    | 0.0124    | 0.0083                 | 0.0039                 |
| Feather     | 0.0034    | 0.0061    | 0.0024                 | 0.0066                 |
| Beehive     | 0.0058    | 0.0032    | 0.0036                 | 0.1933                 |

**FIGURE 13.** The qualitative comparison result of ground truth edge image (left) and result by the proposed algorithm using converted color image from gray (right) (a) orange (b) beehive (c) trashcan (d) stairs.
V. CONCLUSION

When applying the Canny algorithm, users must set three values of high threshold, low threshold, and smoothing window size parameters appropriate for an image. This paper proposed a method to automatically select the Canny algorithm's high and low threshold values with deep reinforcement learning. We present the state, action, and policy network required to transform the problem into deep reinforcement learning. We have proposed three methods where one method belongs to multi-step and two methods belong to a single-step. Also, we proposed a reward model using CNN, which produces an edge quality. The proposed algorithm can cope with new environments by retraining using only images, while supervised learning-based algorithms require ground truth labels. Unsupervised methods show a good performance without requiring training using ground-truth labels. But, their performance on diverse datasets requires further studies. Experimental results using diverse datasets provide the feasibility of the proposed algorithm. However, the reward model used in the proposed algorithm is based on supervised learning. We will focus on overcoming this limitation in future studies.

REFERENCES

1. J.F. Canny, “A computational approach to edge detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 34-43, June 1986.

2. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529-533, Feb. 2015.

3. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra et al., “Playing atari with deep reinforcement learning,” arXiv:1312.5602, 2013.

4. D.R. Martin, C.C. Fowlkes, and J. Malik, “Learning to Detect Natural Image Boundaries Using Local Brightness, Color and Texture Cues,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 26, no. 5, pp. 530-549, May 2004.

5. S. Koniši, A. L. Yuille, M. J. Coughlan, and S. C. Zhu, “Statistical edge detection: Learning and evaluating edge cues,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 1, pp. 57-74, 2003.

6. P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 5, pp. 898-916, 2011.

7. P. Dollar, Z. Tu, and S. Belongie, “Supervised learning of edges and object boundaries,” in IEEE Conference on Computer Vision and Pattern Recognition, 2006, pp. 1964-1971.

8. X. Ren, “Multi-scale improves boundary detection in natural images,” in European Conference on Computer Vision, 2008, pp. 533-545.

9. J. J. Lim, C. L. Zitnick, and P. Dollár, “Sketch tokens: a learned mid-level representation for contour and object detection,” in IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 3158-3165.

10. P. Dollár and C. L. Zitnick, “Fast edge detection using structured forests,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 8, pp. 1558-1570, 2015.

11. W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang, “Deepcontour: A deep convolutional feature learned by positive sharing loss for contour detection,” in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3982-3991.

12. G. Bertusiaux, J. Shi, and L. Torresani, “Deepedge: a multi-scale bifurcated deep network for top-down contour detection,” in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 4380-4389.

13. S. Xie and Z. Tu., “Holistically-nested edge detection,” in IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1395-1403.

14. D. Xu, W. Ouyang, X. Alameda-Pineda, E. Ricci, X. Wang, and N. Sebe, “Learning deep structured multi-scale features using attention-gated crfs for contour prediction,” in Conference on Neural Information Processing Systems, 2017, pp. 3964-3973.

15. J. He, S. Zhang, M. Yang, Y. Shan, and T. Huang, “Bi-directional cascade network for perceptual edge detection,” in IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 3828-3837.

16. J. W. Lu, J. C. Ren and Y. Lu, “A modified Canny algorithm for detecting sky-sea line in infrared images”, in Proc. IS&T/JE, 2006, vol.2, pp.16-18.

17. M. Fang, G. X. Yue and Q. C. Yu, “The study on an application of Otusi method in Canny operator”, in Proc. ISIP’09, 2009, pp. 109-112.

18. N. Otsu, “Threshold selection method from gray-level histograms,” IEEE Trans. on System, Man, and Cybernetics, vol.9, no.1, pp.62-66, 1979.

19. Y.-K. Huo, G. Wei, Y.-D. Zhang, and L.-N. Wu, “An adaptive threshold for the Canny Operator of edge detection,” in International Conference on Image Analysis and Signal Processing, 2010, pp. 371-374.

20. X. Lu, J. Yao, K. Li, and L. Li, “Cannylines: a parameter-free line segment detector,” in IEEE International Conference on Image Processing, 2015, pp. 507-511.

21. R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 2nd ed., Cambridge, MA, USA: MIT Press, 2018.

22. Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436-444, 2015.

23. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneervelsham, M. Lanctot et al., “Mastering the game of go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, 2016.

24. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., “Mastering the game of go without human knowledge,” Nature, vol. 550, no. 7676, 2017.

25. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” Science, vol. 362, no. 6419, pp. 1140-1144, 2018.

26. J. C. Caececo and S. Lazebnik, “Active object localization with deep reinforcement learning,” in International Conference on Computer Vision, 2015, pp. 2488-2496.

27. Z. He, X. Liang, J. Feng, X. Jin, W. Lu, and S. Yan, “Free-structured reinforcement learning for fine-grained object localization,” in Advances in Neural Information Processing Systems, 2016, pp. 127–135.

28. S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Young Choi, “Action-decision networks for visual tracking with deep reinforcement learning,” in IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2711-2720.

29. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv:1409.1556, 2014.

30. K. H. Choi and J. E. Ha, “Edge detection based on U-Net using edge classification CNN,” Journal of Institute of Control, Robotics and Systems, vol. 25, no. 8, pp. 684-689, 2019 (in Korean).

31. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770-778.

32. F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, M. Madhavan, and T. Darrel, “BDD100K: A diverse driving database for heterogeneous multitask learning,” arXiv:1805.04687, 2018.

33. D. P. Kingma, J. Ba, “Adam: A method for stochastic optimization,” arXiv:1412.6980, 2014.

34. N. Goyette, P.-M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, “A novel video dataset for change detection benchmarking,” IEEE Transactions on Image Processing, vol. 23, no. 11, pp. 4663-4679, Nov. 2014.

35. Y. Zitzhaky and E. Peli, “A method for objective edge detection evaluation and detector parameter selection,” IEEE Transactions on...
36. R. Medina-Carnicer, A. Carmona-Poyato, R. Munoz-Salinas and F. J. Madrid-Cuevas, “Determining hysteresis thresholds for edge detection by combining the advantages and disadvantages of threshold methods,” IEEE Transactions on Image Processing, vol. 19, no. 1, pp. 165-173, 2010.

37. R. Medina-Carnicer, R. Munoz-Salinas, E. Yeguas-Bolivar and L. Diaz-Mas, “A novel method to look for the hysteresis thresholds for the Canny edge detector,” Pattern Recognition, vol. 44, pp. 1201-1211, 2011.

38. M. D. Heath, S. Sarkar, T. Sanocki and K. Boyer, “A robust visual method for assessing the relative performance of edge-detection algorithms,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 12, pp. 1338-1359, 1997.

39. A. J. Baddeley, “An error metric for binary images,” in Robust Computer Vision: Quality of Vision Algorithms, Wichmann Verlag Karlsruhe, 1992, pp. 59-78.

40. R. Medina-Carnicer, F. J. Madrid-Cuevas, R. Munoz-Salinas and A. Carmona-Poyato, “Solving the process of hysteresis without determining the optimal thresholds,” Pattern Recognition, vol. 43, pp. 1124-1232, 2010.

41. P. Arbelaez, J. Pont-Tuset, J. Barron, F. Marques and J. Malik, “Multi-scale combinatorial grouping,” in IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 328-335.

42. L.-C. Chen, J. T. Barron, G. Papandreou, K. Murphy and A. L. Yuille, “Semantic image segmentation with task-specific edge detection using CNNs and a discriminatively trained domain transform,” in IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4545-4554.

43. M. Modava and G. Akbarizadeh, “Coastline extraction from SAR images using spatial fuzzy clustering and the active contour method,” International Journal of Remote Sensing, vol. 38, no. 2, pp. 355-370, 2017.

44. T. Hermosilla, J. Palomar-Vazquez, a. Balaguer-Beser and J. Balsabarreiro, “Using street based metrics to characterize urban topologies,” Computers, Environments and Urban Systems, vol. 44, pp. 68-79, 2014.