Study on Stage Generation for 2D Action Game based on DC-GAN

Kotaro NAGAHIRO\textsuperscript{a}, Sho OOI\textsuperscript{b}, Mutsuo SANO\textsuperscript{c}

\textsuperscript{a} Graduate School of Information Science and Technology, Osaka Institute of Technology  
\textsuperscript{b} Faculty of Information Science and Engineering, Ritsumeikan University  
\textsuperscript{c} Faculty of Information Science and Technology, Osaka Institute of Technology

\textsuperscript{a}m1m19a26@st.oit.ac.jp, \textsuperscript{b}SHO.OOI@outlook.jp, \textsuperscript{c}mutsuo.sano@oit.ac.jp

Abstract

Recently, research on generative adversarial networks (GANs) in deep learning has advanced rapidly. For instance, in the field of image recognition, using a GAN, the number of training data was increased. GAN have also been used to create new similar images using training images, which can help designers such as car designer, character designer, game designer and more. In this study, we consider a method to automatically generate a game stage based on a dot-picture using GAN. However, the images generated by GAN using the original game stages are blurred. To solve this problem, we perform preprocessing to the original images. Specifically, we painted out colors for each type of object, and deleted backgrounds and unnecessary objects. In our experiments, we automatically generated game stages using DC-GAN for two 2D action games with different scroll directions (vertical or horizontal). As a result, the proposed method can generate game stages clearly in some game systems.

Keywords: Deep learning, Generative adversarial network, 2D action game, Automatic stage generation, Image generation.

1. Introduction

Recently, research on generative adversarial networks (GANs) in deep learning has advanced rapidly\textsuperscript{(1)}. In the field of image recognition, researchers increased the number of training images using data augmentation \textsuperscript{(2)}. In other words, a GAN can generate a new image similar to a training image, thus addressing the issue of a lack of data. In addition, GAN-based methods have been applied in various ways to support a designer. For example, Reed et al. \textsuperscript{(3)} have studied the generated method of an image from text, Yan et al. \textsuperscript{(4)} have studied the generated of a face image from attribute words. In previous researches\textsuperscript{(5-10)}, they have studied the generation new image from the learned images.

We focus on a method to develop game stage for a game. For the generation of a game stage, in general, game designers generate some game stages during game development and players play a game on the stage made by the designers. Recently, players have developed game stages using the "Super Mario Maker" which runs on Wii U software developed by the Nintendo Corporation. The players then play a game on the stage created by other people. However, games are developed with multiple game stages, and the development cost is expensive. Furthermore, when a user creates a game stage, the process is time consuming and restricted owing to constraints imposed by the game software. In a study of automatic generation of the game stage using GAN, Radford et al. \textsuperscript{(11)} have been studying a game stage of "DOOM" in the first-person shooter (FPS) game. In this study, they have conducted an automatic generation of the 3D game stage using the image of the viewpoint seen from directly above.

With the aim to reduce development costs, we have been studying a method to automatically generate game stages using a GAN. In this study, we focus on a dot-picture 2D scroll game. Specifically, we generate game stages using deep convolution GAN (DC-GAN)\textsuperscript{(12)}; however, when a game stage is generated by DC-GAN using the original image, as shown in figure 1, some problems such as collapse of the game object and noise in the background are observed.

To solve the above problems, we replace some objects in the original images to simple colors (RGB), delete unnecessary color information, and learn the DC-GAN using a pre-processed image \textsuperscript{(13-14)}. Thus, the proposed method was able to generate a stage of a horizontally scrolling game with less noise. In this research, we aim to create multiple game
2. Generating method of a game stage

2.1 DC-GAN

There are several types of GAN using a convolution neural network (CNN), such as the Laplacian pyramid of adversarial networks (LAP-GAN). The LAP-GAN was proposed by Denton et al.\(^{(15)}\); it can create a high-resolution image. However, the LAP-GAN needs multiple networks and processing time. Therefore, we use the DC-GAN proposed by Radford et al.\(^{(12)}\), which can generate high-resolution images without using multiple networks. Figure 2 shows the DC-GAN network. DC-GAN has four features. One is that it does not use a pooling layer; the second is that it does not use a fully connected layer, the third is that it uses batch normalization, and finally, it uses the leaky rectified linear unit (Leaky ReLU).

2.2 Method for increasing learning images

GANs require many learning images for training; however, each game has a limited number of game stages. Hence, we propose a method of dividing a long game stage by constant sliding, as shown in figure 3, thus increasing the number of learning images.

A 2D action game has a long vertical or horizontal stage. Therefore, we slide vertically or horizontally to divide the game stage. In this study, the window size is 256 px × 224 px, and the sliding window size is 16 px, which is one slide of a dot picture. Figure 3 shows an example of a sliding windows using "Super Mario Brothers."

2.3 Issues experienced with GAN using dot-picture

For learning dot pictures, we target nine game stages (1-1, 2-1, 4-1, 5-1, 5-2, 7-1, 8-1, 8-2 and 8-3), all of which have stages with a ground and blue background, from the game "Super Mario Brothers", and increased the number of learning images to 2,080 using the previously described method. In other words, we do not target underground game stages, sea stages, and sky stages. We then performed learning using the DC-GAN. The result is as shown in figure 1. Learning is performed for a total of 100 epochs. Figure 4 shows the generator loss and the discriminator loss for each epoch.

For learning using the dot-picture of original game stages, as shown in figure 1, some problems such as collapsing of the game object and noise in the background are observed. The issues shown in figure 5 are described...
below:
(a) Blurring the game object
The generated game object appears translucent. In this case, the motion of the game object becomes unclear.
(b) Collapsing the game object
The game object of the dot-picture is a quadrangle. However, the game object is generated incorrectly by the DC-GAN and causes gameplay issues.
(c) Noising the background
The background of this game is blue with dots of different colors in some places. These noises may affect the game.

To solve the above problems, in this research, we focus on training images before learning by preprocessing.

2.4 Proposed the method of preprocessing
To solve the aforementioned problems, we replace some objects in the original images with pure colors (RGB) and delete unnecessary color information. The purpose of this pre-processing is to simplify objects and backgrounds and to reduce the image blur and noise generated by the DC-GAN. Furthermore, we can easily modify the deformation of the object. Table 1 shows the conversion results of preprocessing in "Super Mario Bros." The specific process consists of three steps:

(i) Classification of game objects
We classify the objects in the game into three types of "enemy," "block," and "item" and map them to "R," "G," and "B" in terms of color information in RGB, respectively.
(ii) Quantization
We subdivide the behavior of each of the three types of objects and assign numbers between 0 and 255, equally spaced.
(iii) Translation
The value assigned to the game objects is defined as a value on the RGB color scale. However, game objects not related to the player character are converted to white. For instance, the background is replaced white.

Figure 6 shows stage 1-1 in "Super Mario Brothers" after preprocessing. Figure 7 shows a stage of “Kid Icarus” after preprocessing.

3. Experiments
In our experiments, we automatically generated game stages using the proposed method and DC-GAN for two 2D action games with different scroll directions (vertical or horizontal).
In the experiments of this study, we automatically generate game stages using the proposed method and DC-GAN for two 2D action games with different scroll directions (vertical or horizontal). The training count is 100 epochs. The specifications of the computer used are as follows: 2.5 GHz Intel Core i5 CPU, NVIDIA GeForce GTX 860M GPU with 4 GB memory, and 16 GB RAM.

3.1 Game stage of horizontally scroll

In this study, we chose "Super Mario Brothers" for horizontal scrolling, because the numbers of enemy characters and items are few in this game, and stage data of some stages is large. In this experiment, we selected 13 stages of the ground type (1-1, 2-1, 3-1, 3-2, 4-1, 5-1, 5-2, 6-1, 6-2, 7-1, 8-1, 8-2 and 8-3). To increase the number of images, the sliding window was set horizontal, and the system generated 2,762 training images. For preprocessing, the system performed conversions according to the rules listed in Table 1.

3.2 Game stage of vertically scroll

We chose "Kid Icarus" for vertical scrolling, because the location of enemy characters and items is widespread, and stage data of some stages is large. In this experiment, we selected 13 stages of the ground type (1-1, 1-2, 1-3, 3-1, 3-2 and 3-3). To increase the number of images, the sliding window was set vertical, and the system generated 1,227 training images. For preprocessing, the system performed conversions according to the rules listed in Table 2.

4. Result

We described the output results of “Super Mario Brothers” and “Kid Icarus” using DC-GAN. In this study, we set the number of learning times to 100 epochs.

4.1 Result of horizontally scrolling game

Figure 8 shows the generator loss and the discriminator loss, and figure 9 shows a part of the generated game stage using DC-GAN. We describe the generated “Super Mario Brothers” stages using DC-GAN. Figure 8 shows the generator loss and the discriminator loss, and figure 9 shows a part of the generated game stage using DC-GAN. When we observed the generated image, the system
reduced the noise and generated the stage. In other words, the pre-processed game stage resulted in the more stable generation than without pre-processing. When we compare figure 8 and figure 2, the pre-processed method yields lower loss and better learning accuracy.

4.2 Result of vertically scrolling game

Figure 10 shows the generator loss and the discriminator loss, and Figure 11 shows a part of the generated game stage using the DC-GAN. When we compare Figure 9 and Figure 2, both graphs show equal loss, resulting in poor learning accuracy. For this reason, the system generated the same game stages as shown in figure 11. In other words, and the system failed to learn the game stages.

5. Discussion

5.1 Discussion of horizontally scrolling game

In the pre-processed method, the noise of the object and the background is small. The reason for this is believed to be a result of the image being composed of simple information.

Next, focusing on the noise on the image, in figure 1, the noise was sparse; however, in figure 9, the noise occurred at regular intervals. The reason for this is that the object size is 16 px; hence, we think that noise occurred every 16 px.

Comparing figure 4 and figure 8, the proposed method is found to yield lower loss and generate images with better learning accuracy. However, blocks that cannot be cleared by the player actions and stages that cannot be cleared were also generated, thus warranting further improvement.

5.2 Discussion of vertically scrolling game

Although the game generation method for the vertically scrolling game is the same as that of the horizontally scrolling game, the system failed the game stage generation. For this reason, we consider the difference between game systems to be the cause of failure. The game targeted in this research has fewer commonalities in the arrangement of objects on the screen, as shown in figure 12. Specifically, in the case of the horizontally scrolling game, there is always a block in the lower part of the screen because the player character's traveling direction is right and there is gravity. On the other hand, in the vertically scrolling game, the player character's travels upwards, and blocks are arranged in various ways, for instance blocks for climbing upwards are arranged as stepping stones. For this reason, we think that the
system failed in learning how stages are to be developed.

To solve this problem, we think that it is necessary to learn at each stage with commonality or to increase the number of learning images.

6. Conclusion

In this study, performed stage-generation for a 2D action game using a DC-GAN. However, when the system learns using a standard image, problems such as increased noise were observed. Therefore, we proposed a method to replace game objects with specific rules as pre-processing. As a result, the proposed system could eliminate the noise; however, the result was that system could not learn the vertically scrolling game system.

In the future, we will focus more on the stage generation of a vertically scrolling game. Furthermore, we will implement the game stage obtained via pre-processing as a standard game stage so that players can attempt this stage.

References


