Instance Segmentation Method of User Interface Component of Games

Shinjin Kang and Jong-in Choi

1 School of Games, Hongik University, Sejong 30016, Korea; directx@hongik.ac.kr
2 Department of Digital Media Design and Applications, Seoul Women’s University, Seoul 01797, Korea

Correspondence: funtech@swu.ac.kr

Received: 22 August 2020; Accepted: 14 September 2020; Published: 17 September 2020

Abstract: On the game screen, the UI interface provides key information for game play. A vision deep learning network exploits pure pixel information in the screen. Apart from this, if we separately extract the information provided by the UI interface and use it as an additional input value, we can enhance the learning efficiency of deep learning networks. To this end, by effectively segmenting UI interface components such as buttons, image icons, and gauge bars on the game screen, we should be able to separately analyze only the relevant images. In this paper, we propose a methodology that segments UI components in a game by using synthetic game images created on a game engine. We developed a tool that approximately detected the UI areas of an image in games on the game screen and generated a large amount of synthetic labeling data through this. By training this data on a Pix2Pix, we applied UI segmentation. The network trained in this way can segment the UI areas of the target game regardless of the position of the corresponding UI components. Our methodology can help analyze the game screen without applying data augmentation to the game screen. It can also help vision researchers who should extract semantic information from game image data.

Keywords: instance segmentation; UI segmentation; game user interface

1. Introduction

With the recent advancement of deep learning technology, various deep learning technologies are influencing game development. The GAN algorithm adopted in level generation has been applied on a pilot basis to a large number of dungeons and level generation [1–3]. Convolution neural network (CNN) or parameter-based reinforcement learning algorithms applied to nonplayable characters (NPC) are used to develop NPC that can learn user behavior patterns [4–6]. Learning algorithms for time-series data based on long-short term memory (LSTM) used in traditional user log analysis are applied to technologies such as user churn prediction [7,8] or bot detection [9]. Natural language recognition technologies used in the voice interface provide users with a new interface experience [10].

In order for deep learning technology to be used in the game field, a large amount of data must be secured. Game data can be accessed freely by a game company which developed a game, but it is not easily accessible from outside for researchers who use this data. Therefore, game-related researchers in the deep learning community attempted to get access to the relevant parameters through various indirect methods at the client level [11]. However, this indirect data acquisition process has been a major obstacle for deep learning researchers to conduct research in the game field. Because of this technical obstacle, datasets in the game field are smaller than datasets in the image processing field in terms of the size and number. Moreover, this has become the biggest obstacle for image-based algorithms to be applied to the game research field.

In this paper, in order to address these issues, we propose a segmentation method that extracts only the meaningful UI information from the game play screen. We built a large amount of UI
Image segmentation data required for machine learning by using the game engine and the published UI image dataset. After this, using this data, we designed a network that segmented only the UI components on the general game play screen and trained it. When the trained network was applied to games, it could detect the main UI components on the game screen at the level of approximately 80%. Our methodology demonstrates that UI component images can be effectively extracted from the game play screen without additional data labeling. These segmented images have the advantage of enabling meaningful game play analysis with only the game screen by facilitating additional semantic analysis later on.

2. Previous Works

2.1. Application Cases of CNN-Based Algorithms in the Game Field

CNN-based algorithms have been actively introduced in the game field. Ref. [12] trained convolutional neural networks to play Go by training them to predict the way that professional Go players move Go stones and then recognizing the patterns of Go like humans. Ref. [13] presented a CNN, despite being relatively slow, for evaluating the state of real-time strategy (RTS) games that goes beyond the commonly used material-based evaluation by taking into account the spatial relationship between units. Ref. [14] learned through iterative self-play using a CNN-based learning model that can effectively learn patterns in three different games and improved the understanding of the games by training on the results of its previous work without sophisticated domain knowledge. Ref. [15] retrained the final layer of the well-known Inception V3 architecture developed by Google to find out a classification technique that could recognize traditional Bangladeshi sports. The result showed an average accuracy of approximately 80% in correctly recognizing five sporting events. Ref. [16] presented a novel approach to learning a forward simulation model through simple search for pixel input. They significantly improved the method of predicting future states compared to the base CNN and trained a game play agent by applying the trained model. Ref. [17] presented an effective architecture, a multisize convolution neural network (MSCNN), to improve the accuracy of winner prediction in RTS games. MSCNN can capture functions for different game states using various filter sizes. Ref. [18] presented an innovative deep learning approach to ball identification in tennis images. By making the most of the potential of a convolutional neural network classifier to decide whether a tennis ball was observed in a single frame, they overcame a typical problem that could occur in long video sequences. Ref. [19] presented the first deep learning model that successfully learned direct control through high-dimensional input using reinforcement learning. This model is a trained convolutional neural network, a variant of Q-learning, where input was raw pixels and output was a value function that estimated future rewards. This method surpassed all previous approaches in Atari’s six games and outperformed a human expert in three games. Ref. [20] presented a new approach to player modeling based on a convolutional neural network trained on game event logs. Using the Super Mario game, they demonstrated high accuracy in predicting various measures of player experience. Unlike [19,21] which presented the first architecture dealing with 3D environments in first-person shooter games that were partially observable. They dramatically improved the training speed and performance of game agents by presenting a method of augmenting these models to make the most of game function information such as the presence of enemies or items in the training phase.

In the game field, CNN algorithms have been used in various fields. All are for extracting feature parameters, and the entire game screen image has generally been processed as an input value. This can be applied naturally to learning classic games without UI information in the game. However, this ultimately prolongs the process of searching for meaningful parameters within the game image and increases the number of related learning parameters. If the core region of interest (ROI) can be reduced, the image-based learning efficiency can be greatly improved.
2.2. Segmentation Methods

Instance segmentation is a crucial, complex, and challenging area in research on machine vision. With the aim of predicting object class labels and object instance masks for each pixel, it localizes various object instance classes that are present in various images. With the advent of convolutional neural networks (CNN), various instance segmentation frameworks have been proposed [22–24]. They rapidly increased the segmentation accuracy [25]. Mask R-CNN [22] is a simple and efficient instance segmentation approach. Fully convolutional networks (FCN) that took the initiative in Fast/Faster R-CNN [26,27] were used to predict segmentation masks along with box regression and object classification. For high performance, a feature pyramid network (FPN) [28] was used to extract step-by-step network functions and has a semantically powerful function using a top-down network path having lateral connections. The instance segmentation can be largely divided into the following four methods: (1) classification of mask proposals, (2) detection followed by segmentation, (3) labeling pixels followed by clustering, and (4) dense sliding window methods.

Classification of mask proposals was gradually transformed from the classical method to a new method with an efficient structure such as RCNN [29]. Despite the better segmentation accuracy, there were some issues with RCNN-related technologies. For example, it is based on a multistage pipeline that is slow and hard to optimize since each stage must be learned separately. As the functions for each proposal should be extracted from the entire image of CNN, issues arose such as storing, time, and scale of detection. Later on, RCNN was followed by Fast RCNN [26] and Faster RCNN [27], which solved these issues. One of the most successful technologies related to detection followed by segmentation is Mask RCNN [22]. Mask RCNN extends Faster R-CNN [27] detection algorithm by employing a relatively simple mask predictor. Mask RCNN is simple to train, better generalized, and can be implemented by adding a little computational overhead to Faster R-CNN. An instance segmentation approach based on Mask R-CNN [30–32] has recently shown promising results in the instance segmentation problem. Another approach to instance segmentation, labeling pixels followed by clustering [24,33,34], uses the technology generated for semantic segmentation [35,36]. This approach includes labeling by category of all image pixels. The pixels are then grouped into object instances using a clustering algorithm. This approach benefits from the positive development of modern semantic segmentation that can predict high-resolution object masks. Compared to detection followed by segmentation, labeling pixels followed by clustering is less accurate in frequently used benchmarks. Generally, more computational power is required because of the powerful computation capability required for pixel labeling. DeepMask [37,38], InstanceFCN [39], etc., use CNN to generate mask proposal by dense sliding window methods. TensorMask [40] improves performance by using a different architecture compared to class agnostic mask generation techniques. In addition, unlike DeepMask and InstanceFCN, TensorMask includes classification for several classes performed in parallel with the prediction mask. This function is effective in instance segmentation.

Especially, the segmentation method in the face recognition had a great influence on the face recognition and generation method. DeepLabv3+, a model proposed in [41], improved segmentation results along object boundaries by extending DeepLabv3 through adding a simple and effective decoder module. In addition, they produced a faster and more powerful encoder-decoder network by further exploring the Xception model and applying separable convolution by depth to both Atrous Spatial Pyramid Pooling and decoder modules. In International Conference on Multimedia Retrieval (ICMR) in 2015, Ref. [42] published a face detector based on the basic AlexNet. The 2015 algorithm calculated the results by fine-tuning the AlexNet trained using ImageNet with face images. However, the detection performance was not that good then, thus the final decision was made using SVM at the final stage. In 2015, Ref. [43] proposed a deep learning-based face detector that determined the final face by separately detecting facial parts such as head, eyes, nose, mouth, and beard. In particular, related technology was proposed to detect faces with severe occlusion, which used a total of five facial part detectors consisting of a total of seven layers, and the results from them were used to calculate the final result. In CVPR 2015, Ref. [44] published a paper in which the cascade scheme, which was
widely used in the conventional face detectors, was connected in series to a network that was not deep in order to solve a speed issue. This study first removes the nonface area by using a small face size and increases the resolution once a face candidate is selected later on. Furthermore, since the network is not deep, it can speed up execution without the help of a GPU. Ref. [45] proposed a state-of-the-art and real-time object detection system that was able to detect more than 9000 object categories. It runs significantly fast while outperforming state-of-the-art methods like Faster RCNN.

The following attempts have been made to increase the learning efficiency of segmentation within an image using game data. It was mainly conducted in the research of autonomous vehicles. Since creating large datasets with pixel-level labels is very costly owing to the amount of human resources required, Ref. [11] presented a method to speedily generate accurate semantic label maps in pixel units for images extracted from modern computer games. It validated the presented method by creating dense pixel-level semantic annotations for 25,000 images synthesized by a realistic open-world computer game. Ref. [46] used a direct perception method in order to estimate the affordance for driving action, which was different to two traditional paradigms used in the autonomous driving methods, that is, a mediated perception method that analyzed the entire scene to make a driving decision and a behavior reflection method that directly mapped an input image to driving action by a regressor. For this, it proposed mapping an input image to a few key perception indicators directly related to the affordance of a road/traffic state for driving. It provided a set of compact but complete descriptions for the scene so that a simple controller could drive autonomously. To prove this, they trained deep convolutional neural networks using 12 h of driving history in a video game and showed that it could operate well to drive a car in a wide range of virtual environments. Ref. [47] provided a publicly available visual surveillance simulation test bed based on a commercial game engine called object video virtual video (OVVV). It simulated multiple synchronized video streams in various camera configurations, including static and omnidirectional cameras, in a virtual environment filled with humans and vehicles controlled by a computer or a player. Ref. [48] presented a method that the pedestrian appearance model learned in a virtual scenario could successfully operate to detect pedestrians of real images. For this, an appearance-based pedestrian classifier was learned using HOG and linear SVM after recording the training sequence in a virtual scenario. Ref. [49] has the same purpose as [48]. For this, they designed V-AYLA, a domain adaptation framework. By collecting a few pedestrian samples from the target domain (i.e., real-world) and combining them with multiple examples from the source domain (i.e., virtual-world), they tested various techniques to train a domain adaptive pedestrian classifier that would run in the target domain.

Our study is to apply object segmentation methods to the game play screen. We aim to segment UI components first that contain the most meaningful information in the game. Our research was influenced by [20], which obtained data from game play data, and [11], which proposed the methodology of detecting game objects.

3. Methods

3.1. Automatic UI Clipping and Paired Synthetic UI Data Generation

For this experiment, we generated synthetic UI Data using Unity, a commercial game engine. To automatically generate synthetic UI Data, we used two images captured from the game screen. By comparing the two captured images by pixel, we eliminated the background of the game by removing the part with different colors and leaving the part with the same color, and then extracted only the UI. For the comparison of the color of a pixel, we compared the corresponding pixel with the four pixels around it (top, bottom, left, right). This is because there is a slight color difference depending on the encoding method when an image is saved. For all pixels, colors were compared with the proportion of 60% of the central color and 40% of the surrounding color. If the hue value differs by more than 10% for two pixels, they are regarded as different colors. If the color of a pixel at
the coordinates \(x, y\) in the 2D image is \(C(x, y)\), the color \(C_p(x, y)\) of the pixel for comparison can be calculated as follows:

\[
C_p(x, y) = 0.5 \times C(x, y) + 0.1 \times \{C(x + 1, y) + C(x - 1, y) + C(x, y + 1) + C(x, y - 1)\}
\]  

We used RGB (red, green, blue) colors for the comparison of colors. We set the range of RGB colors to a value between 0 and 1, respectively, and the sum of the difference values of each color was used as a basis for comparison. Figure 1 shows two screen images used for extracting UIs.

![Figure 1. Two screen images captured for UI extraction.](image)

The set of pixels extracted in this way is classified into several groups. To find the agglomerated pixels, we used the flood fill algorithm. This is an effective algorithm, which starts with a single pixel and finds all the pixels connected to it. We regarded a pixel as connected if it was adjacent to any of the four directions of the top, bottom, left, and right. We find all groups of the agglomerated pixels by scanning all pixels starting from the 2D coordinate \((0,0)\) of the image and to the coordinate \((\text{width}-1, \text{height}-1)\). In this experiment, in this way, we automatically extracted all UI images contained in the screenshot. We only extracted UIs larger than the pixel size of \(20 \times 20\) to eliminate noise. In addition, to generate paired data, we produced a monochromatic UI image painted with one color corresponding to the original UI image. For instance, if 10 UI images are extracted, 10 monochromatic UI images corresponding to them are automatically generated. Figure 2 shows the UIs extracted in this way and the monochrome processed UIs. On the left are the extracted UI images, and on the right are the monochrome processed UI images.

![Figure 2. Extracted UIs (left), monochrome processed UIs (right).](image)

To assign colors to numerous UI images, we created basic colors and applied them in sequence. Assuming that the darkest color was 0 and the brightest color was 1, we created 15 basic colors that could be distinguished by applying 0.5 unit to each of the R (red), G (green), and B (blue) colors. It could be divided into smaller units such as 0.25. However, since it was difficult to identify colors with the naked eye, we divided them in 0.5 units. Table 1 shows the basic color used in this experiment. If the number of UIs is more than 15, we applied the colors again from the beginning. That is, the color of the 16th UI becomes color 0.
Table 1. Basic colors for monochrome UI.

| no. | Color(R, G, B) | no. | Color(R, G, B) | no. | Color(R, G, B) |
|-----|----------------|-----|----------------|-----|----------------|
| 1   | (0.0, 0.0, 0.0) | 6   | (0.0, 1.0, 1.0) | 11  | (0.0, 0.5, 0.0) |
| 2   | (1.0, 1.0, 1.0) | 7   | (1.0, 0.0, 1.0) | 12  | (0.0, 0.0, 0.5) |
| 3   | (1.0, 0.0, 0.0) | 8   | (1.0, 1.0, 0.0) | 13  | (0.0, 0.5, 0.5) |
| 4   | (0.0, 1.0, 0.0) | 9   | (0.5, 0.5, 0.5) | 14  | (0.5, 0.0, 0.0) |
| 5   | (0.0, 0.0, 1.0) | 10  | (0.5, 0.0, 0.0) | 15  | (0.5, 0.5, 0.5) |

Next, we capture a few play screens that hide the UI from the game screen. General PC games provide a function to hide the UI to allow nice screenshots to be taken. If it is a mobile game without such a function, it is fine to capture and use the area excluding the UI. For the automatic generation of synthetic UI Data, we randomly select one of the captured images where the UI is excluded as the background and randomly place the extracted original UI images over the background. Here, we processed all UIs not to overlap each other. To this end, we created a box shaped collision area in all UIs, and in case they overlap each other, they are regarded as collision and rearranged. This process is iterated until all UIs do not overlap. To smoothly avoid collision, we give priority to large UIs. In other words, the largest UI is arranged first, the next largest UI second, and the smallest UI last. For the calculation of the area, we used the box collision area in each UI. A single image data is generated by capturing the screen in a state where all UIs do not overlap. Next, after replacing all the original UI images with the corresponding monochrome UI images over the same background, other paired image data is generated by capturing the screen. We can obtain the desired number of paired synthetic UI data by iterating this process. Figure 3 shows the paired synthetic UI data generated in this way. On the left is an image where the extracted UIs are randomly arranged, and on the right is an image where the monochrome processed UIs are arranged in the same position.

![Figure 3. Paired synthetic UI Data.](image)

To use it as an experimental dataset, we connected the left and right images of Figure 4 by reducing them to the size of 256 × 256. Figure 4 shows the final generated experimental data, and the total size of the image is 512 × 256. In this study, we used this experimental data by automatically generating it, and it took approximately 5 min to generate 10,000 images. There might be cases where auto labeling does not work well with our tool. These are the cases where the outline of icons is unclear or the alpha value is entered too much. Figure 5 shows these cases. In this case, coloring should be done directly on the relevant area through a separate manual operation.
3.2. Network

With the paired synthetic data generated above, when receiving the general game screen as an input value, we infer the UI area from it and segment this part. We used the Pix2Pix framework that showed quality performance in paired data learning as the baseline network. In the network, when the general game screen is inputted as a network input value, the generator produces a segmented image of each UI area as an output value. In Pix2Pix, the generator is a convolutional network with U-net architecture. Since Pix2Pix has the same input and output resolution, and has the characteristic of maintaining the detail and shape to a degree, information loss occurs when the encoder-decoder structure is used. This can be reduced by using skip-connection. Here, the game screen, which is an input image, and the segmented UI image, are paired up with each other and exist in the form of a $512 \times 256$ size image. After the generator, the synthetic image is concatenated with the target color image. Therefore, the number of color channels will be six ($height \times width \times 6$). This is fed as input to the discriminator network. In Pix2Pix, patchGAN is used as a discriminator. The patchGAN network takes the concatenated input images and produces an output that $N \times N$. 

**Figure 4.** Automatically generated experimental data.

**Figure 5.** Images of cases where auto labeling does not work well.
The objective function of the Pix2Pix GAN is as follows:

\[
V(G, D) = \min_G \max_D \mathbb{E}_y[\log(D(y))] + \mathbb{E}_x[\log(1 - D(G(x)))] + \mathbb{E}_{xy}[\| y - G(x) \|_1] \tag{2}
\]

Here, \(y\) refers to the real image of the target domain and \(x\) refers to the real image of the source domain. The first and second terms are called the adversarial loss, and it is characterized by using the loss employed the GAN.

\[
\mathbb{E}_y[\log(D(y))] + \mathbb{E}_x[\log(1 - D(G(x)))] \tag{3}
\]

In other words, that the real \(y\) is real, and \(G(x)\), the \(y\) mapped from \(x\), is fake, is the Pix2Pix discriminator loss. On the contrary, that \(G(x)\) is real is the generator loss. The last term is the reconstruction Loss, which is the loss employed in the conventional CNN-based learning method.

\[
\mathbb{E}_{xy}[\| y - G(x) \|_1] \tag{4}
\]

The difference between the real \(y\) and \(G(x)\) made from \(x\) is minimized. Figure 6 shows the Pix2Pix GAN structure we used. Figure 7 shows the generator and discriminator network structure. Figure 8 shows the results of our network performance experiment. We experimentally have the best visual results when we removed the last encoder and first decoder of the original Pix2Pix generator. The generator’s optimization objective function includes an L1 loss, which has a direction to minimize the Euclidean distance between the original image and the generated image. Therefore, there is a tendency to focus on the average value of the image, that is, low frequency. Since the L1 loss is in charge of the low frequency region, the discriminator focuses on high frequency to determine whether it is real or fake. The discriminator’s patch size should be determined in consideration of the overall image size and the relationship between pixels. In this experiment, when a size of 70 × 70 was used, a clear image could be obtained.

Figure 6. Proposed GAN architecture.
Figure 7. Proposed generator and discriminator architecture.

Figure 8. Network performance experiment results. (left: original Pix2Pix generator with 16 × 16 patch Size, middle: original Pix2Pix generator with 80 × 80 patch size, right: modified Pix2Pix generator with 70 × 70 patch size).

4. Experimental Results

To verify the effectiveness of this system, we tested it in three games (Kingdom Rush [50], Blade and Soul [51], and Black Desert [52]). The three games have provided a long-term stable game service with great commercial success in the mobile and PC game fields. We chose these three games because their UI design was representative of the genre. UI complexity differs in these three games, thus we aim to compare how well the UI is extracted in each complexity. We first made a simple screen
capture program and captured screen shots of 20,000 images for each game in a situation where three games were running. Next, UIs of the corresponding screenshot of each game were automatically auto-labeled through the automatic UI extractor we proposed. The processing time is approximately 10 min for each game. Overall, since each game arranges similar UI components adjacent to each other, we can confirm that similar UI components are clustered and auto-labeled. The segmentation number for each game is 7 for Kingdom Rush, 30 for Blade and Soul, and 37 for Black Desert. This number can be interpreted as the number of UI groups that are geographically isolated from each other by the floor fill algorithm in the tool we use.

Training was done for each game. All images are resized to $256 \times 256$ for training. For optimization, we set the max-number of iterations to 200, the learning rate as 0.001, and its decay rate as 20% per five iterations. We use the SGD optimizer for training with the batch size = 8. NVIDIA RTX Titan GPU took approximately two days to perform a single training. The images resulting from creating segmentation images with the test set of each game using the trained network model are shown in Figures 9–11, respectively. During the experiment, the position and background of UI components were randomly arranged to prevent overfitting at a specific position.

Figure 9 shows the results of applying our system to Kingdom Rush. This is a casual tower defense game in which the core interaction is to place a small number of designated buildings on the map every turn. Since only designated buildings are placed, the UI functions actually required are quite limited. Furthermore, since the touch interface is used for this game, the UI icons are spaced over a certain distance to prevent double touches. It is easy to detect each separate UI component with our tool. The system automatically generated synthetic data segmented into seven groups and trained the network through this. When applying the data trained in this way to the original Kingdom Rush, we could accurately detect the corresponding UI area with a pixel accuracy of 88%.

![Figure 9. Result of Kingdom Rush UI segmentation.](image)

Figure 10 shows the results of applying our system to PC MMORPG Blade and Soul. Blade and Soul is an action-based PC MMORPG game. Quite complex types of information are displayed on the screen, and players interact by learning them. Therefore, unlike Kingdom Rush mentioned
above, the chat window, map screen, and character information UI are further displayed on the screen. Moreover, the space between the UI components is very small because the mouse, which allows fine control, is used as the basic interface. In this experiment, we attempted to confirm whether the system could automatically detect these complex UI components. The system automatically generated synthetic data segmented into 30 groups and trained the network through this. When applying the data trained in this way to the original Blade and Soul, we could accurately detect the corresponding UI area with a pixel accuracy of 85%. The network seems to learn stably although the number of the UI components to be detected is larger than that of Kingdom Rush. Unlike Kingdom Rush, the UI components of Blade and Soul are quite small and have high image resolution. Therefore, if the corresponding UI component has a similar color and shape to the background image, it may be difficult for the network to detect it. However, our network demonstrates its capability to segment the UI components.

Figure 10. Result of Blade and Soul UI segmentation.

Figure 11 shows the results of applying our system to another PC MMORPG game, Black Desert. Black Desert is a skill-based PC MMORPG game. As in Blade and Soul, quite complex types of information are displayed on the screen, and players interact by learning them. Unlike Blade and Soul, Black Desert is a skill-based game, thus a large number of skills are exposed on the UI. In this experiment, we attempted to confirm whether the system could automatically detect these skills. As a skill-based game, a number of skill shortcuts can be placed around the characters in the center, thus the UI positional characteristics are not apparent. The system automatically generated synthetic data segmented into 37 groups and trained the network through this. When applying the data trained in this way to the original Black Desert, we could accurately detect the corresponding UI area with a pixel accuracy of 84%. In this experiment, the number of the UI components to be detected was more than that of Blade and Soul, which led to smaller UI components. The network seems to learn stably, though. The experimental results show that the performance of our system decreases with an increase in the number of UIs for segmentation, and at the image level of $256 \times 256$, the relatively stable UI segmentation is expected to be achieved up to approximately 40.
Table 2 shows the overall segmentation performance of our network. First, we reduced the size of UI screens of the three games to $256 \times 256$, then checked the number of segmented UI components, the size of the most basic UI button, and the approximate spacing between each UI component and defined approximate complexity from this. We measured pixel accuracy, mean accuracy, mean IU, and frequency weighted IU values for Kingdom Rush, Blade and Soul, and Black Desert. Mean accuracy values for them were 0.820, 0.797, and 0.777, respectively. It is confirmed that, overall, pixel accuracy is reduced, being inversely proportional to the number of UI component classes we intend to classify. Since Kingdom Rush, a mobile game, is based on a touch-based UI, our system could automatically extract all UI components. However, as for Blade and Soul and Black Desert, MMORPG games, the number of the UI components detected by the system is approximately 50% below of the actual number of UI icons. This is because the UI components are subdivided into skill units. Therefore, our system shows that UI segmentation is more effectively achieved relatively in mobile games.

Table 2. Evaluation metric for images produced by the generator for each game.

|                          | Kingdom Rush | Blade and Soul | Black Desert |
|--------------------------|--------------|----------------|--------------|
| Number of Segmented UI Components | 7            | 30             | 37           |
| Size of a Single Button (Approx. Pixel) | Large ($25 \times 25$) | Small ($5 \times 5$) | Small ($5 \times 5$) |
| Spacing between UI Components (Approx. Pixel) | Long (5 pixel) | Short (1–2 pixel) | Short (1–2 pixel) |
| Overall Complexity       | Low          | High           | High         |
| Pixel Accuracy           | 0.882        | 0.851          | 0.842        |
| Mean Accuracy            | 0.820        | 0.797          | 0.777        |
| Mean IU                  | 0.707        | 0.665          | 0.671        |
| Frequency Weighted IU    | 0.793        | 0.734          | 0.791        |

Figure 12 shows the loss graph during training. Red indicates Kingdom Rush, pale green Blade and Soul (BnS), and dark green Black Desert. Only Black Desert is shown to have relative difficulty in learning. Overall, learning was stabilized at around 200 epochs. With an increase in the number of the UI classes to be classified, the loss of the generator drops.
Figures 13 and 14 show the images resulting from segmentation tests with the game screen image of another game of the same genre using the trained network model. In this experiment, under the assumption that UI configuration and design would be similar for the same genre, we attempted to confirm how much similar UI components could be detected. Figure 13 shows the results when the network trained with Kingdom Rush is applied to Iron Marine [53] of a similar genre. Compared to the ground truth result images in red, it is confirmed that, overall, the position and configuration are detected to a degree. However, it cannot detect the central part of the UI components well. Pixel accuracy is approximately 67%. Figure 14 shows the results of the network trained with Black Desert when using World of Warcraft [54] as an input value. Similar to the above case, overall, the network detects the UI components with a pixel accuracy of 63%. However, there are cases where it incorrectly determines some of the background image as UI. This experiment shows the potential that our network can also detect some of the UIs of similar genres. Since our networks are trained for each game, when applied to different game genres, they did not show high accuracy due to overfitting to target game. For the universal UI detection, accuracy is expected to increase if all images of the relevant genres are collected and learned.

![Figure 12. Variations in loss graph during training.](image)

![Figure 13. Result of applying the UI segmentation network trained with Kingdom Rush to Iron Marine. (Left: Original image, Middle: Ground truth, Right: Segmented image).](image)
5. Conclusions

In this study, we proposed a method for automatically segmenting UI components on the game screen using a Pix2Pix. In addition, we proposed a novel methodology that could mass-produce paired datasets required for deep learning research by using the synthetic data. Our method can reduce data labeling cost by developing a tool that automatically extracts and labels only the UI area from the game screen for supervised learning. The UI segmentation network trained in this way has the advantage of being able to detect UI components in the games regardless of their position and it can detect UI areas in other similar games. Our study can be used to extract useful information from UIs in game vision-based deep learning research. Our system can obtain various metadata, such as HP, skill, and chat, which can be taken from UI areas within the game screen. The data obtained in this way can be annotated back to the corresponding images and used to create additional image-parameter pair data. These datasets can then be used for various derivative studies. Our study can be effective as a preprocessing tool for various systems that exploit supervised learning data and facilitate research on vision-based game deep learning.

Our system has a number of limitations. First, for images with high complexity in the original image, UI components can be segmented into groups because our automatic UI extraction tool does not work normally. Second, the general UI assumes a square shape in numerous cases, so the difference from the background image is clear, but the difference in shape between UI components is negligible. Therefore, there may be limitations in recognizing the difference between each UI component. To solve this problem, the segmentation network should be trained to focus on textures rather than shapes. Finally, our experiment was only tested on the genre of mobile games based on touch interfaces and PC MMORPG games. There is a need to apply it to a wider variety of genres and games to check its stability.

Our system has the potential to develop in two aspects in the future. The first is to extract the actual string by applying the text classifier to the extracted segmentation image. This makes it possible to generate more precise metadata. The second is to develop universal UI network segmentation that can extract all game UIs within a similar genre, not a segmentation network specialized for one game. Consequently, the usefulness of the network will increase significantly. To achieve this, it will be necessary to design and combine a multidomain segmentation network.

**Author Contributions:** Methodology, S.K.; software, J.-i.C. All authors have read and agreed to the published version of the manuscript.
**Funding:** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No.NRF-2019R1A2C1002525) and a research grant from Seoul Women’s University (2020-0392).

**Conflicts of Interest:** The authors declare no conflict of interest.

**References**

1. Giacomello, E.; Lanzi, P.L.; Loiacono, D. DOOM level generation using generative adversarial networks. In Proceedings of the 2018 IEEE Games, Entertainment, Media Conference (GEM), Galway, Ireland, 15–17 August 2018; pp. 316–323.

2. Volz, V.; Schrum, J.; Liu, J.; Lucas, S.M.; Smith, A.; Risi, S. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In Proceedings of the Genetic and Evolutionary Computation Conference, Kyoto, Japan, 15–19 July 2018; pp. 221–228.

3. Awiszus, M.; Schubert, F.; Rosenhahn, B. TOAD-GAN: Coherent Style Level Generation from a Single Example. *arXiv* 2020, arXiv:2008.01531.

4. Torrado, R.R.; Bontrager, P.; Togelius, J.; Liu, J.; Perez-Liebana, D. Deep reinforcement learning for general video game ai. In Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG), Maastricht, The Netherlands, 14–17 August 2018; pp. 1–8.

5. Nadiger, C.; Kumar, A.; Abdelhak, S. Federated reinforcement learning for fast personalization. In Proceedings of the 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), Sardinia, Italy, 3–5 June 2019; pp. 123–127.

6. Glavin, F.G.; Madden, M.G. DRE-Bot: A hierarchical First Person Shooter bot using multiple Sarsa (λ) reinforcement learners. In Proceedings of the 2012 17th International Conference on Computer Games (CGAMES), Louisville, KY, USA, 30 July–1 August 2012; pp. 148–152.

7. Lee, E.; Jang, Y.; Yoon, D.M.; Jeon, J.; Yang, S.I.; Lee, S.K.; Kim, D.W.; Chen, P.P.; Guitart, A.; Bertens, P.; et al. Game data mining competition on churn prediction and survival analysis using commercial game log data. *IEEE Trans. Games* 2018, 11, 215–226. [CrossRef]

8. Yang, W.; Huang, T.; Zeng, J.; Yang, G.; Cai, J.; Chen, L.; Mishra, S.; Liu, Y.E. Mining Player In-game Time Spending Regularity for Churn Prediction in Free Online Games. In Proceedings of the 2019 IEEE Conference on Games (CoG), London, UK, 20–23 August 2019; pp. 1–8.

9. Lee, J.; Lim, J.; Cho, W.; Kim, H.K. In-game action sequence analysis for game bot detection on the big data analysis platform. In Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Singapore, 10–12 November 2014; Springer: Cham, Switzerland, 2015; Volume 2, pp. 403–414.

10. Zagal, J.P.; Tomuro, N.; Shepitsen, A. Natural language processing in game studies research: An overview. *Simul. Gaming* 2012, 43, 356–373. [CrossRef]

11. Richter, S.R.; Vineet, V.; Roth, S.; Koltun, V. Playing for data: Ground truth from computer games. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016; pp. 102–118.

12. Clark, C.; Storkey, A. Training deep convolutional neural networks to play go. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1766–1774.

13. Stanescu, M.; Barriga, N.A.; Hess, A.; Buro, M. Evaluating real-time strategy game states using convolutional neural networks. In Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games (CIG), Santorini Island, Greece, 20–24 September 2016; pp. 1–7.

14. Yakovenko, N.; Cao, L.; Raffel, C.; Fan, J. Poker-CNN: A pattern learning strategy for making draws and bets in poker games using convolutional networks. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016.

15. Islam, M.S.; Foyals, F.A.; Neehal, N.; Karim, E.; Hossain, S.A. InceptB: A CNN based classification approach for recognizing traditional bengali games. *Procedia Comput. Sci.* 2018, 143, 595–602. [CrossRef]

16. Guzdial, M.; Li, B.; Riedl, M.O. Game Engine Learning from Video. In Proceedings of the IJCAI, Melbourne, Australia, 19 August 2017; pp. 3707–3713.

17. Huang, J.; Yang, W. A multi-size convolution neural network for RTS games winner prediction. In Proceedings of the MATEC Web of Conferences, Phuket, Thailand, 4–7 July 2018; EDP Sciences: Les Ulis, France, 2018; Volume 232, p. 01054.
18. Reno, V.; Mosca, N.; Marani, R.; Nitti, M.; D’Orazio, T.; Stella, E. Convolutional neural networks based ball detection in tennis games. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Salt Lake City, UT, USA, 18–22 June 2018; pp. 1758–1764.

19. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. arXiv 2013, arXiv:1312.5602.

20. Liao, N.; Guzdial, M.; Riedl, M. Deep convolutional player modeling on log and level data. In Proceedings of the 12th International Conference on the Foundations of Digital Games, Raleigh, NC, USA, 29 May–1 June 2017; pp. 1–4.

21. Lample, G.; Chaplot, D.S. Playing FPS games with deep reinforcement learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 23 June 2017.

22. Li, Y.; Qi, H.; Dai, J.; Ji, X.; Wei, Y. Fully convolutional instance-aware semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Venice, Italy, 22–29 October 2017; pp. 2359–2367.

23. Bai, M.; Urtasun, R. Deep watershed transform for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Venice, Italy, 22–29 October 2017; pp. 5221–5229.

24. Everingham, M.; Van Gool, L.; Williams, C.K.; Winn, J.; Zisserman, A. The pascal visual object classes (voc) challenge. Int. J. Comput. Vis. 2010, 88, 303–338. [CrossRef]

25. Girshick, R. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1440–1448.

26. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2117–2125.

27. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587.

28. Liu, S.; Qi, L.; Qin, H.; Shi, J.; Jia, J. Path aggregation network for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 8759–8768.

29. Peng, C.; Xiao, T.; Li, Z.; Jiang, Y.; Zhang, X.; Jia, K.; Yu, G.; Sun, J. Megdet: A large mini-batch object detector. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 6181–6189.

30. Chen, K.; Pang, J.; Wang, J.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Shi, J.; Ouyang, W. Hybrid task cascade for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 4974–4983.

31. Kirillov, A.; Levinek, E.; Andres, B.; Savchynskyy, B.; Rother, C. Instancecut: From edges to instances with multicut. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Venice, Italy, 22–29 October 2017; pp. 5008–5017.

32. Arnab, A.; Torr, P.H. Pixelwise instance segmentation with a dynamically instantiated network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Venice, Italy, 22–29 October 2017; pp. 441–450.

33. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3431–3440.

34. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. Semantic image segmentation with deep convolutional nets and fully connected crfs. arXiv 2014, arXiv:1412.7062.

35. Pinheiro, P.O.; Collobert, R.; Dollár, P. Learning to segment object candidates. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 1990–1998.
38. Pinheiro, P.O.; Lin, T.Y.; Collobert, R.; Dollár, P. Learning to refine object segments. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016; pp. 75–91.

39. Dai, J.; He, K.; Li, Y.; Ren, S.; Sun, J. Instance-sensitive fully convolutional networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016; pp. 534–549.

40. Chen, X.; Girshick, R.; He, K.; Dollár, P. Tensormask: A foundation for dense object segmentation. In Proceedings of the IEEE International Conference on Computer Vision, Long Beach, CA, USA, 15–20 June 2019; pp. 2061–2069.

41. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European Conference on Computer Vision (ECCV), Salt Lake City, UT, USA, 18–22 June 2018; pp. 801–818.

42. Farfade, S.S.; Saberian, M.J.; Li, L.J. Multi-view face detection using deep convolutional neural networks. In Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, Shanghai, China, 23–26 June 2015; pp. 643–650.

43. Yang, S.; Luo, P.; Loy, C.C.; Tang, X. From facial parts responses to face detection: A deep learning approach. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 3676–3684.

44. Li, H.; Lin, Z.; Shen, X.; Br, T.J.; Hua, G. A convolutional neural network cascade for face detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Santiago, Chile, 7–13 December 2015; pp. 5325–5334.

45. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Venice, Italy, 22–29 October 2017; pp. 7263–7271.

46. Chen, C.; Seff, A.; Kornhauser, A.; Xiao, J. Deepdriving: Learning affordance for direct perception in autonomous driving. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 2722–2730.

47. Taylor, G.R.; Chosak, A.J.; Brewer, P.C. Ovvv: Using virtual worlds to design and evaluate surveillance systems. In Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, MN, USA, 17–22 June 2007; pp. 1–8.

48. Marin, J.; Vázquez, D.; Gerónimo, D.; López, A.M. Learning appearance in virtual scenarios for pedestrian detection. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA, 13–18 June 2010; pp. 137–144.

49. Vazquez, D.; Lopez, A.M.; Marin, J.; Ponsa, D.; Gerónimo, D. Virtual and real world adaptation for pedestrian detection. IEEE Trans. Pattern Anal. Mach. Intell. 2013, 36, 797–809. [CrossRef] [PubMed]

50. Kingdom Rush, Ironhide Game Studio. Available online: https://android.kingdomrush&hl=en (accessed on 22 August 2020).

51. Blade and Soul, NCsoft. Available online: https://www.bladeandsoul.com/en/ (accessed on 22 August 2020).

52. Black Desert, Perl Abyss. Available online: https://www.blackdesertonline.com/hashashin (accessed on 22 August 2020).

53. Iron Marine, Ironhide Game Studio. Available online: https://android.ironmarines (accessed on 22 August 2020).

54. World of Warcraft, Blizzard. Available online: https://worldofwarcraft.com/ (accessed on 22 August 2020).