Make Your Own Sprites: Aliasing-Aware and Cell-Controllable Pixelization

ZONGWEI WU, South China University of Technology, China
LIANGYU CHAI, South China University of Technology, China
NANXUAN ZHAO, University of Bath, United Kingdom
BAILIN DENG, Cardiff University, United Kingdom
YONGTUO LIU, University of Amsterdam, Netherlands and South China University of Technology, China
QIANG WEN, Hong Kong University of Science and Technology, China and South China University of Technology, China
JUNLE WANG, Tencent, China
SHENGFENG HE*, South China University of Technology, China

Fig. 1. We propose a novel pixelization approach that can automatically and robustly convert icons, clip arts, paintings, posters, and game scenes to pixel arts. The resulting pixel arts contain crisp and sharp details (see the portrait in the upper left corner and the icons below), while being controllable in the cell size (see the upper right corner). (© Tencent, © Nintendo Co., Ltd., © Paul Lamy de La Chapelle.)

Pixel art is a unique art style with the appearance of low resolution images. In this paper, we propose a data-driven pixelization method that can produce sharp and crisp cell effects with controllable cell sizes. Our approach overcomes the limitation of existing learning-based methods in cell size control by introducing a reference pixel art to explicitly regularize the cell

*Corresponding author.

Authors' addresses: Zongwei Wu, South China University of Technology, Guangzhou, China, zongweiwu999@gmail.com; Liangyu Chai, South China University of Technology, Guangzhou, China, icepoint1018@gmail.com; Nanxuan Zhao, University of Bath, United Kingdom, nanxuanzhao@gmail.com; Bailin Deng, Cardiff University, Cardiff, United Kingdom, DengB3@cardiff.ac.uk; Yongtuo Liu, University of Amsterdam, Amsterdam, Netherlands and South China University of Technology, Guangzhou, China, y.liu6@uva.nl; Qiang Wen, Hong Kong University of Science and Technology, Hong Kong, China and South China University of Technology, Guangzhou, China, chineseqiangwen@gmail.com; Junle Wang, Tencent, Shenzhen, China, jljunlewang@tencent.com; Shengfeng He, South China University of Technology, Guangzhou, China, hesfe@scut.edu.cn.

0730-0301/2022/12-ART193 $15.00
https://doi.org/10.1145/3550454.3555482

structure. In particular, the cell structure features of the reference pixel art are used as an auxiliary input for the pixelization process, and for measuring the style similarity between the generated result and the reference pixel art. Furthermore, we disentangle the pixelization process into specific cell-aware and aliasing-aware stages, mitigating the ambiguities in joint learning of cell size, aliasing effect, and color assignment. To train our model, we construct a dedicated pixel art dataset and augment it with different cell sizes and different degrees of anti-aliasing effects. Extensive experiments demonstrate its superior performance over state-of-the-arts in terms of cell sharpness and perceptual expressiveness. We also show promising results of video game pixelization for the first time. Code and dataset are available at https://github.com/WuZongWei6/Pixelization.

CCS Concepts: • **Applied computing → Fine arts**; • **Computing methodologies → Image processing**.

Additional Key Words and Phrases: Pixelization, Generative Adversarial Networks, Image-to-Image Translation

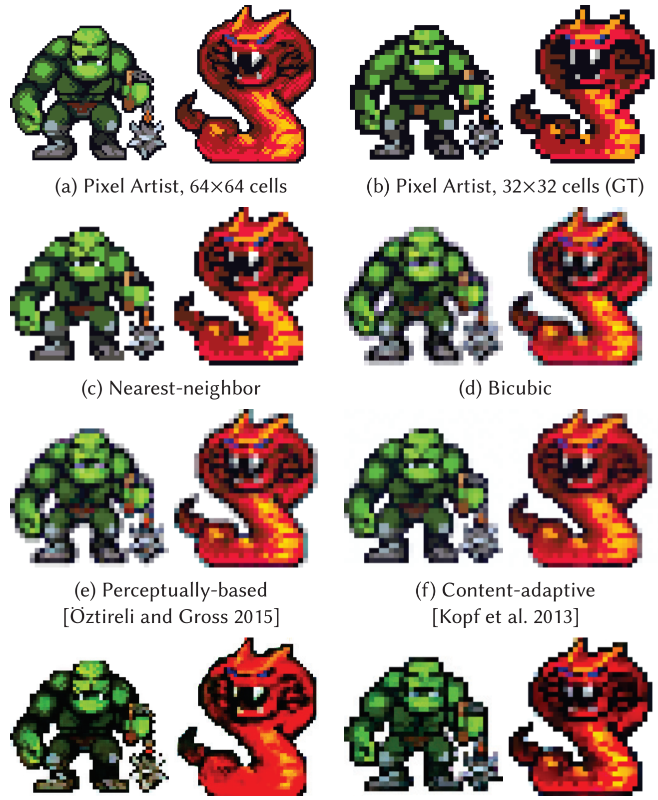
ACM Reference Format:
Zongwei Wu, Liangyu Chai, Nanxuan Zhao, Bailin Deng, Yongtuo Liu, Qiang Wen, Junle Wang, and Shengfeng He. 2022. Make Your Own Sprites: Aliasing-Aware and Cell-Controllable Pixelization. *ACM Trans. Graph.* 41, 6, Article 193 (December 2022), 16 pages. https://doi.org/10.1145/3550454.3555482

1 INTRODUCTION

Pixel art is an art form with the appearance of low resolution images. It can be considered as a 2D grid where each grid cell contains N×N pixels of the same color. It was primarily associated with the art style of video games from the 80s. Because of its unique style and nostalgia, pixel art has gained popularity not only in video games but also in other domains such as advertisement. Due to the limited resolution, pixel art needs careful arrangement of the colored cells to effectively represent the underlying content, which requires artistic sense and graphic design skills and can be time-consuming even for professional artists. Therefore, there is a demand for computational tools that can automatically convert a higher-resolution image into pixel art. Such a tool should ideally meet the following requirements:

- *Aliasing-aware*: the pixel art should have sharp and crisp edges instead of anti-aliasing appearance (see Fig. 2 (d) for an example of such anti-aliasing appearance). This helps to imitate the style of early video games that pixel arts originate from, where the small palette limits the possibility of applying anti-aliasing.
- *Cell-controllable*: the user can control the cell size of the result as needed (see the upper right part of Fig. 1).
- *Detail-preserving*: the result should preserve the appearance of features and continuous edges as much as possible.
- *Unpaired-data*: for a learning-based method, its training should not require paired data of pixel arts and their corresponding high-resolution images, since large-scale datasets of such paired images are currently not available.

Existing methods do not meet all criteria simultaneously. For example, a simple approach to converting images into pixel arts is image downsampling. However, classical downsampling approaches such as nearest-neighbor and bicubic often produce unsatisfactory results: the former can lose crucial details from the original image (see the discontinuous block silhouette curves in Fig. 2 (c)), while the latter can produce undesirable anti-aliasing appearance (see Fig. 2 (d)). More sophisticated approaches such as [Kopf et al. 2013] and [Öztireli and Gross 2015] attempt to better preserve the content and details. They do not require training data and can generate small images of arbitrary resolution, thus being cell-controllable. However, their use of adaptive filters or optimized colors still introduces anti-aliasing effects (see Fig. 2 (e) & (f)), hence they are not aliasing-aware and defy the aesthetic sense of pixel art.



(a) Pixel Artist, 64×64 cells (b) Pixel Artist, 32×32 cells (GT)

(c) Nearest-neighbor (d) Bicubic

(e) Perceptually-based [Öztireli and Gross 2015] (f) Content-adaptive [Kopf et al. 2013]

(g) Deep Pixelization [Han et al. 2018] (h) Ours

Fig. 2. Comparison between different pixelization methods. (a) & (b) show two images created by pixel artists for the same underlying content, (© Silver Lemur Games and © Krzysztof Koźmik), where (b) has a lower resolution of cells and a larger cell size, and (a) is an upscaled version of (b)[1]. (c) to (h) show results from different methods using (a) as input to generate a pixel art with the same cell resolution as (b). Classical downscaling methods ((c) & (d)) either lose details (e.g., the discontinuous black silhouette curves in (c)) or introduce anti-aliasing effects and blurry edges. The content-adaptive method from [Kopf et al. 2013] and the perceptually-based method from [Öztireli and Gross 2015] preserve the content and details better, but still introduce anti-aliasing effects. The deep-learning-based image-to-image translation method from [Han et al. 2018] leads to non-uniform cell sizes and distorted colors. Our learning-based method generates a result with faithful colors and the desired cell size, without anti-aliasing artifacts.

To preserve semantically salient features, Han et al. [2018] propose the first data-driven approach to pixel art generation from non-pixel art images. Different from downsampling the input image, it

[1]https://www.silverlemurgames.com/2020/07/29/the-secret-story-behind-pixel-sizes/

formulates a cycled image-to-image translation process between the input image and the resulting pixel art in an unsupervised manner, to overcome the barrier of missing paired data. Although it has made breakthroughs in aliasing-awareness and detail-preservation, there are still two main issues. First, its data organization and network design do not take the cell size into account; as a result, although it can achieve an appearance similar to pixel art, the result may contain non-uniform cells in different sizes (see Fig. 2 (g)), i.e., it is not cell-controllable. In addition, its model entangles the content and the pixelization style, which may result in distorted colors.

In this paper, we propose a deep learning-based method for aliasing-aware and cell-controllable pixel art generation that overcomes the limitations of existing methods. Our method generates a result image with the same resolution as the input, where the pixels form cells with the designated cell size. To control the resulting cell size, we treat different cell sizes as different image styles, and adopt the idea of style transfer [Huang and Belongie 2017] to decouple the image content and the style. Specifically, we introduce a cell size code that is determined by applying an encoder network module (CSEnc) on a reference pixel art with the intended cell size. The cell size code modulates the kernel weights of the intermediate convolutional layers for an image translation network module (I2PNet), to guide the conversion from a non-pixel art image to an intermediate result with the designated cell style. As the intermediate result may still suffer from anti-aliasing effects, we feed it into another network module (AliasNet) to recover the final pixel art with the desired aliasing appearance. During training, the generated result and the reference pixel art are also fed into a discriminator module (NumEnc) to ensure they have similar styles and the same cell size. To enable bi-directional and cyclical training, we also introduce a network module (P2INet) to depixelize a pixel art into a non-pixel art image, and enforce cycle consistency during training. To train our model, we collect a Basic Pixel Art Dataset containing pixel arts of different resolutions, and augment it into two datasets for training. These include an Aliasing Dataset that contains pixel arts and their variants with different degrees of anti-aliasing effects, and a Multi-cell Dataset that contains pixel arts with different cell sizes.

Similar to [Han et al. 2018], our method does not require paired training data of non-pixel art images and their corresponding pixel arts. Compared to [Han et al. 2018], our use of the cell size code and a cell-aware discriminator enables our network to generate pixel arts with regular cell appearance and the desired cell size. The novel architecture and the dedicated datasets also help our network to better preserve the input color and avoid the color distortion artifacts from [Han et al. 2018]. Extensive experiments show that our approach can generate pixel arts with crisp and regular cells for different input image styles, and consistently outperforms existing methods by large margins. It also provides the first feasible approach for coherent conversion of a video game into pixel art style.

Our contributions can be summarized as follows:

- We propose the first deep learning-based method that can convert a non-pixel art image into a pixel art with a designated cell size.
- We present a model that disentangles the learning of color appearance and cell structure, which can achieve cell regularity, aliasing appearance, and color fidelity simultaneously.
- We develop a method to augment a basic pixel art dataset into two datasets with different cell sizes and anti-aliasing effects.
- We demonstrate crisp and robust pixelization results over various cell sizes on different types of inputs. In particular, our method provides the first feasible approach to video game pixelization.

2 RELATED WORK

Pixelization. A direct approach to pixelization is to downscale the input image to a lower resolution. Nearest-neighbor, a sampling method, downscales an image by selecting the color value of the nearest point. However, this can lead to loss of important details. An alternative way is to apply linear filters with a fixed kernel such as bicubic, but they often introduce over-smoothing effects. Kopf et al. [2013] adaptively vary the filter kernels in a bilateral way to better focus on local image features and avoid ringing. Öztireli and Gross [2015] optimize the downscaling method according to the perceptual metric SSIM to retain essential features as well as local structures. Weber et al. [2016] use convolutional filters to ensure high frequency details, while Liu et al. [2018] propose to capture salient features with L_0 regularization. However, the above methods follow a locally averaging principle and are not effective in retaining sharp edges in salient regions.

Some works formulate the pixelization task as an optimization problem. In [Inglis and Kaplan 2012] and [Inglis et al. 2013], vector line arts are pixelated by optimizing a rasterization to preserve the shape properties. Gerstner et al. [2013; 2012] present a multi-step iterative algorithm that converts high-resolution images to pixel arts by optimizing a set of superpixels and an associated color palette. Kuo et al. [2016] propose a framework for pixel art animation, by jointly optimizing prominent feature lines of individual frames according to animation quality metrics. Shang and Wong [2021] propose an optimization-based approach that can pixelize portrait images using limited colors, but it fails to distinguish the edges when the colors of the object are similar to the background. Although the above algorithms achieve visually acceptable results, most of them are not fully automatic and cannot provide end-to-end solutions.

Learning-based Methods. Deep learning has received widespread attention in recent years and provided fully automatic solutions to many problems. Using deep learning for end-to-end pixelization is beneficial because manually drawing pixel art or tuning parameters for semi-automatic methods can be time-consuming. The online pixelization project PixelMe [Sato 2020] is built upon pix2pix [Isola et al. 2017], which uses conditional GAN to perform supervised learning on paired data. However, it is difficult to collect such paired datasets for pixel arts. CycleGAN [Zhu et al. 2017] proposes an unsupervised image-to-image translation solution with unpaired data. Following this approach, Han et al. [2018] propose an unsupervised pixelization method that allows training with unpaired data. However, their approach does not provide cell size control and may produce results with distorted colors. Kuang et al. [2021] focus on preserving the contour details based on CycleGAN, at the expense of introducing artifacts. In comparison, our new solution integrates cell controllability, crispness, and robustness simultaneously.

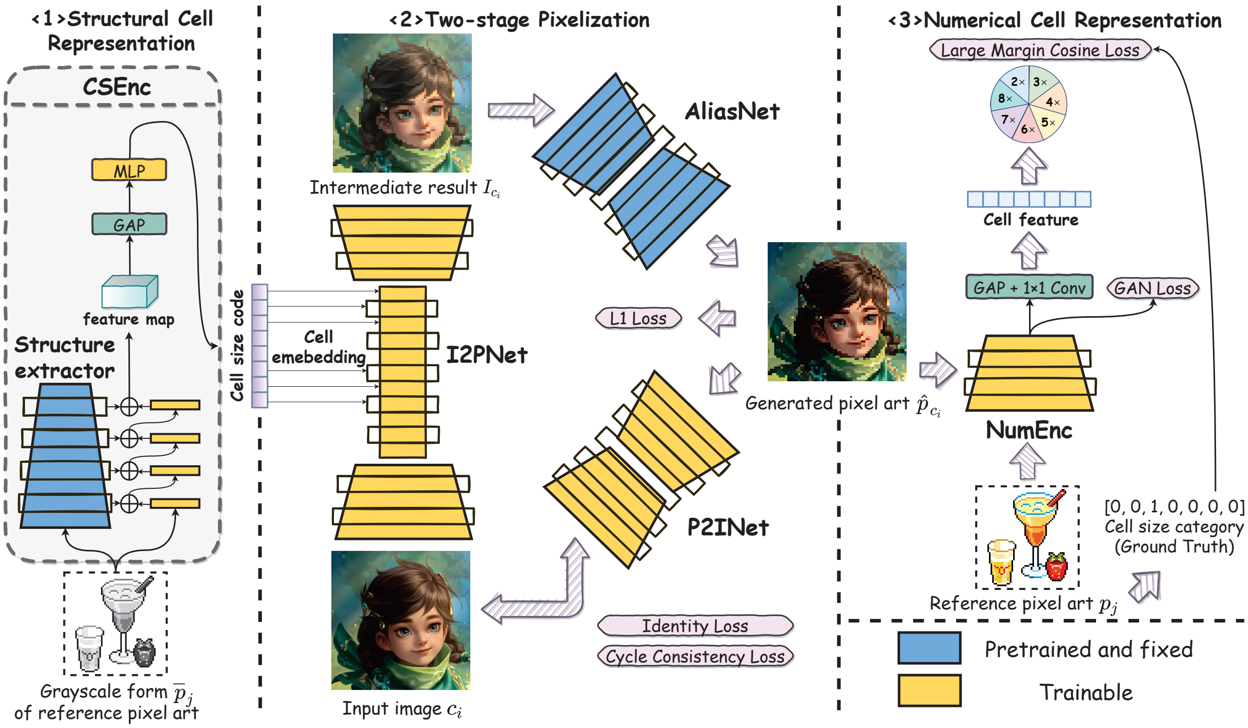


Fig. 3. Overview of the proposed method. To control the cell size of the generated pixel art, during training our method takes a reference pixel art as an additional input, and encodes it into a cell size code using the Cell Structure Encoder (CSEnc) module (<1>). The cell size code is then used in a two-stage pixelization process (<2>): it is first embedded into the I2PNet, takes an input image and produces an intermediate result with the desired cell size; the result is further fed into the AliasNet to remove anti-aliasing effects and obtain a crisp pixel art output. During training, the generated output is depixelized to the original input via the P2INet for retaining the correct cell colors. The Numerical Encoder (NumEnc) is used as a discriminator and enforces the same cell size style between the output and reference pixel arts (<3>). (© eBoyArts, © Tencent.)

3 OVERVIEW

In this paper, we consider a pixel art as a grid of square cells, where each cell contains $N{\times}N$ pixels of the same color. For such a pixel art, we say that it has a cell size $N\times$. In the most simple case, each cell contains only one pixel (i.e., with a cell size 1×), and we refer to such pixel arts as "one cell one pixel". For easier display or editing, such pixel arts can be upscaled using nearest-neighbor interpolation into a form with a cell size $N\times$ where $N > 1$. On the other hand, pixel arts with a cell size $N\times$ ($N > 1$) can also be downscaled to a one-cell-one-pixel form using nearest-neighbor interpolation.

Given an input image, our goal is to convert it into a pixel art with the same pixel resolution and a cell size $N\times$, where the parameter N is specified by the user. In this way, the generated pixel art has an $N\times$ reduction in its effective resolution (i.e., the resolution for its grid of cells), while its pixel resolution remains the same as the input. This setting enables us to design a single network that generates pixel arts with different cell sizes, making the generation process cell-controllable. An alternative approach to cell-controllability would be to train a separate downsampling network for each cell size. Compared to this approach, our single-network design offers a few benefits. First, although pixel arts of various cell sizes are visually different, the underlying low-level features (edges, corners, textures, etc.) are similar. Our single-network approach can facilitate the learning of such common features across different cell sizes and achieve better results. It can also save training time, as there is no need to re-learn the common features for different cell sizes. Finally, a single network can be deployed more easily in practice.

Using image samples $\{c_i\}_{i=1}^{H}$ from the input image domain C and pixel art samples $\{p_j\}_{j=1}^{M}$ from the pixel art domain $\mathcal{P}$, our network learns a mapping F from C to the power set of $\mathcal{P}$, such that an image c is mapped to a set $F(c)$ that contains pixel arts with the same underlying content as c and different cell sizes. Fig. 3 shows an overview of our network architecture.

A particular challenge is that the samples $\{c_i\}$ and $\{p_j\}$ are not paired, since the "original" image of a pixel art is usually not available. To address this issue, during training we use the sample pixel arts $\{p_j\}$ as reference to provide structural information that guides the pixel art generation. Specifically, to control the cell size of the generated pixel art, we introduce a high-dimensional *cell size code* as an auxiliary input. During training, the cell size code is derived by applying a cell size encoder module (CSEnc) to the grayscale form $\overline{p}_j$ of a reference pixel art p_j, which guides the network to generate a pixel art with the same cell size as p_j. The cell size code is utilized as an auxiliary input to an image-to-pixel-art network module (I2PNet), which converts the input image c_i into an intermediate image I_{c_i} that is closer to the desired pixel art but may still

Fig. 4. Examples from our non-pixel art dataset. Note that the first image is from [Zitnick and Parikh 2013], the second one is from [Royer et al. 2020], and others are from © Tencent.

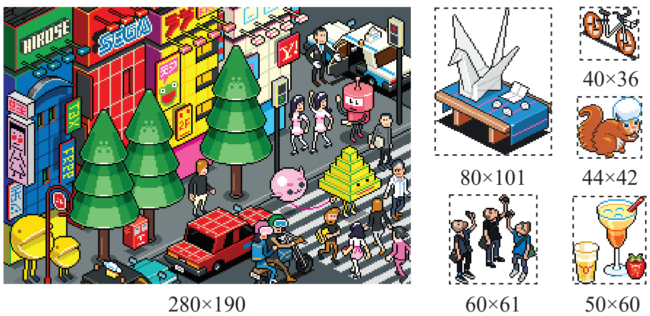


Fig. 5. Examples from our Basic Pixel Art Dataset, which contains images in different resolutions. Note that each cell of these images contains only one pixel. (© eBoyArts.)

suffer from anti-aliasing effects with blurred appearance and color-shifting. The intermediate image is then fed into another module (AliasNet) to remove the anti-aliasing effects and generate the final pixel art $\hat{p}_{c_i}$. During training, we follow CycleGAN [Zhu et al. 2017] and use a pixel-art-to-image network module (P2INet) to convert the result $\hat{p}_{c_i}$ back to the input image, and enforce the cycle consistency between the input image and the resulting pixel art. In addition, we introduce a numerical encoder module (NumEnc) as a discriminator for our GAN training, to ensure the result $\hat{p}_{c_i}$ generated by AliasNet has a pixel art appearance. To further ensure $\hat{p}_{c_i}$ has the desired cell size, we apply NumEnc to $\hat{p}_{c_i}$ and the reference pixel art p_j respectively to classify their cell sizes and penalize the difference.

In the following, we will first present our datasets in Section 4, followed by our network modules and their training in Section 5.

4 DATASETS

The training of our model requires both non-pixel art and pixel art images. For the former type, we collect a dataset consisting of 4235 non-pixel art images, where 790 images are from the CartoonSet [Royer et al. 2020], 277 images are from the Abstract Scene dataset [Zitnick and Parikh 2013], and the rest are from the Internet. All images are scaled to the resolution of 512×512. Fig. 4 shows some images from the dataset.

For pixel art images, we collect and generate three datasets:

- a *Basic Pixel Art Dataset* that contains pixel arts in one-cell-one-pixel forms;
- an *Aliasing Dataset* that includes pixel arts and their variants with different degrees of anti-aliasing effects;
- a *Multi-cell Dataset* of pixel arts with different cell sizes.

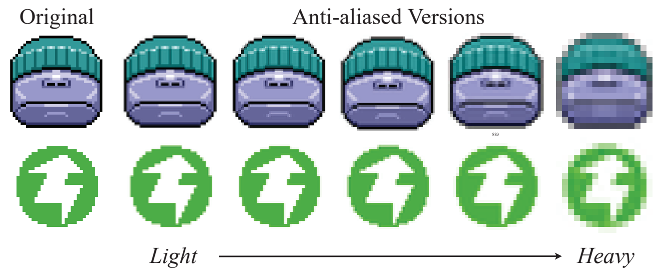


Fig. 6. Examples from our Aliasing Dataset, which contains ground-truth pixel arts (“Original”) and their variants with different degrees of anti-aliasing (“Anti-aliased Versions”). (© eBoyArts.)

The Aliasing Dataset and the Multi-cell Dataset are both derived from the Basic Pixel Art Dataset, as explained below.

Basic Pixel Art Dataset. The Basic Pixel Art Dataset contains a variety of pixel art images in one-cell-one-pixel forms. To construct the dataset, we first collect 4033 pixel art images from the Internet. For images that have a cell size larger than 1×, and we downscale them to a one-cell-one-pixel form using nearest-neighbor downsampling. The final images in the dataset have resolutions between 1847×1701 and 11×9. Some examples are shown in Fig. 5.

Aliasing Dataset. In order to train the AliasNet, we need paired pixel art images with and without anti-aliasing effects in a fixed resolution. To this end, we synthesize anti-aliased versions of the images in the Basic Pixel Art Dataset as follows. We first select pixel arts with a resolution no higher than 128×128, and upscale them to 256×256. Then we apply the Lanczos filter [Turkowski 1990] to downsample these 256×256 images to 80×80, 64×64, 48×48, 32×32, and 16×16, respectively, to introduce anti-aliasing effects. Finally, all the downsampled images are upscaled back to 256×256 using nearest-neighbor interpolation. In this way, we obtain a set of 256×256 pixel arts together with a sequence of images with the same resolution and content but different degrees of anti-aliasing (see Fig. 6 for some examples). In total, we have 2500 pixel art images directly derived from the Basic Pixel Art Dataset, and 12500 images as their anti-aliased versions. These images are used to train the AliasNet for removing anti-aliasing effects.

Multi-cell Dataset. To facilitate multi-batch training and loss function design, we would like the training pixel arts to have a fixed resolution. Since the images in the Basic Pixel Art Dataset have diverse resolutions, they must be scaled to the same resolution first if we want to use them for multi-batch training. However, this may lead to an ambiguity in the cell size of the resulting image, since such scaling may result in a non-integer scaling ratio, or different scaling ratios in the horizontal and vertical directions. In addition, we would like the training pixel arts to cover a variety of cell sizes, to help the network learn the characteristics of various cell sizes. To this end, we use the Basic Pixel Art Dataset to synthesize a Multi-cell Dataset that consists of pixel arts in the same resolution and different cell sizes. From the images in the Basic Pixel Art Dataset, we generate a set of images in the resolutions of 128×128, 86×86, 64×64,

Fig. 7. Examples from our Multi-cell Dataset, which contains pixel arts in different cell sizes. (© eBoyArts.)

52×52, 43×43, 37×37, and 32×32, respectively: for a basic pixel art image smaller than the target resolution, we pad it with pixels of the background color to reach the desired resolution; on the other hand, if a basic pixel art image is larger than the target resolution, we crop non-constant color regions into the desired resolution. The resulting images are then upscaled by a factor of 2×, 3×, 4×, 5×, 6×, 7×, and 8× respectively using nearest-neighbor interpolation, and cropped to the resolution of 256×256. Since the images from the Basic Pixel Art Dataset are all in one-cell-one-pixel forms, such scaling and cropping operations result in pixel arts with a cell size equal to the scaling factor. In this way, we obtain a Multi-cell Dataset with a fixed 256×256 resolution and different cell sizes from 2× to 8×. Our Multi-Cell Dataset consists of 7000 images, with 1000 images for each cell size. Fig. 7 shows some examples from the dataset.

5 OUR APPROACH

To leverage unpaired data, our network is trained in a bi-directional and cyclical way. On the one hand, given a non-pixel art image c_i and a reference pixel art p_j with its grayscale form $\overline{p}_j$, the training follows a data flow of CSEnc→I2PNet→AliasNet→P2INet: a pixel art result $\hat{p}_{c_i}$ is first generated using I2PNet and AliasNet from the image c_i and the cell size code of $\overline{p}_j$, and then restored back to a non-pixel art image c_i' by P2INet. On the other hand, given a reference pixel art p_j and its grayscale form $\overline{p}_j$, the training follows a data flow of P2INet→CSEnc→I2PNet→AliasNet: p_j is first converted by P2INet to a non-pixel art image $\hat{c}_{p_j}$, which is then restored to a pixel art image p_j' using I2PNet and AliasNet with the help of the cell size code of $\overline{p}_j$. The NumEnc module is only used during training as a discriminator for the pixel art output of AliasNet. In this section, we first present the details of our network components, followed by our training strategy and the loss function design.

5.1 Network Structure

Cell Size Encoder (CSEnc). The goal of CSEnc is to extract a feature vector (the cell size code) from the grayscale form $\overline{p}_j$ of the reference pixel art p_j, to be used as an auxiliary input to I2PNet to indicate the desired cell size. To this end, we pre-train a structure extractor module for a cell size classification task, whose layers extract from $\overline{p}_j$ the structural features relevant to the cell size. The structure extractor follows the VGG19 architecture [Simonyan and Zisserman 2015] and is pre-trained with a cross-entropy loss using

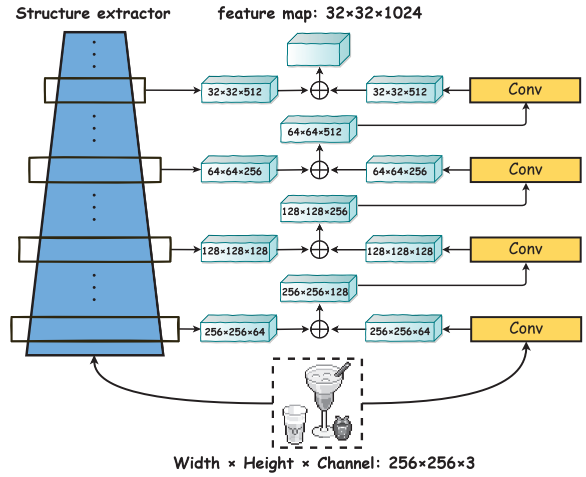


Fig. 8. Extraction of feature map within our CSEnc module. (© eBoyArts.)

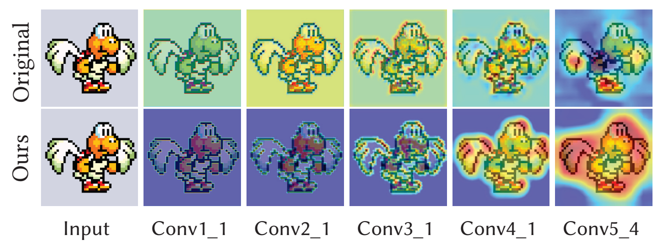


Fig. 9. Class activation maps of original VGG19 pre-trained with ImageNet (top row) and our structure extractor (bottom row). The original VGG19 highlights everywhere with warm colors (especially in the lower convolutional layers), while our extractor can focus on regions of interest and the individual activation point follows a cell structure. The shallow layers of our extractor focus on local regions with activations in smaller cells, and the deeper ones focus more on the global structure. (©Nintendo Co., Ltd.)

our Multi-cell Dataset. We then introduce four trainable convolutional layers, with output dimensions of 256×256×64, 128×128×128, 64×64×256 and 32×32×512, respectively. The output of each layer is concatenated with the output of a corresponding layer from the structure extractor with the same output dimension and then fed to the next layer (see Fig. 8). In this way, the final concatenated output provides a feature map that encapsulates structural features of $\overline{p}_j$ at different scales. We feed this feature map to a global average pooling (GAP) layer followed by a trainable multi-layer perceptron (MLP), to produce a cell size code S of dimension 2048×1.

We train the structure extractor with our Multi-cell Dataset instead of directly using VGG19 pre-trained with ImageNet [Deng et al. 2009], because ImageNet consists of non-pixel art images and is less effective in extracting pixel art features. Fig. 9 shows the class activation maps produced using XGrad-CAM [Fu et al. 2020] for the intermediate layers of our structure extractor, and compares them with those from VGG19 pre-trained with ImageNet. We can see that our structure extractor concentrates on the cell structure at each layer, resulting in an effective cell feature representation.

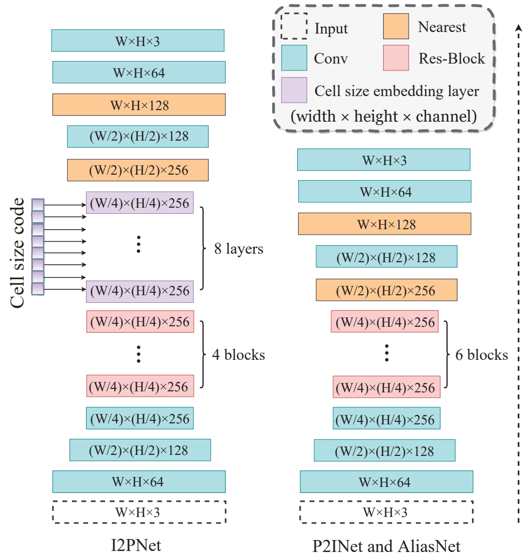


Fig. 10. Architectures of I2PNet, AliasNet and P2INet. The latter two share the same architecture, which only differs from I2PNet in the middle layers.

Image-to-Pixel-Art Network (I2PNet). The I2PNet is an encoder-decoder network (see Fig. 10 left) that generates an intermediate pixelization result from the input image c_i, using the cell size code S from CSEnc as auxiliary guidance for the intended cell size. To incorporate the cell size information into the generation process, we introduce cell size embedding layers to merge S with the intermediate feature map produced by the encoding block of I2PNet. Specifically, we introduce eight intermediate convolutional layers between the encoder block and decoding block, where each layer receives its input in 256 channels. Accordingly, we divide the 2048-dimensional cell size code into eight 256-dimensional blocks s^l $(l = 1, 2, \ldots, 8)$, to match the number of input channels for the intermediate convolutional layers. Following the style modulation approach from StyleGAN2 [Karras et al. 2020], we modify the trainable convolutional kernel weights $\{w^l_{xyz}\}$ at the l-th layer as follows:

$$\overline{w}^l_{xyz} = (s^l_x \cdot w^l_{xyz}) \Bigg/ \sqrt{\sum\nolimits_{x,z} (s^l_x \cdot w^l_{xyz})^2 + \epsilon}\,, \tag{1}$$

where x is the index of the input feature map channels, y denotes the output channel numbers, z represents the spatial position, and ϵ is a small constant to avoid division-by-zero. The weights $\{\overline{w}^l_{xyz}\}$ are then used as the actual kernel weights for the convolutional layers. As a result, the cell size code S modulates the input to the convolutional layers in a way that is independent of the scale of S [Karras et al. 2020].

AliasNet. The AliasNet is an encoder-decoder network (see Fig. 10 right for its structure) that aims to remove the anti-aliasing effect that may be present in the intermediate result generated by I2PNet, which helps to obtain a pixel art image with the desired aliasing appearance. We pre-train AliasNet with the Aliasing Dataset constructed in Section 4, using images with different degrees of anti-aliasing as input, and the original pixel art as the ground truth for the

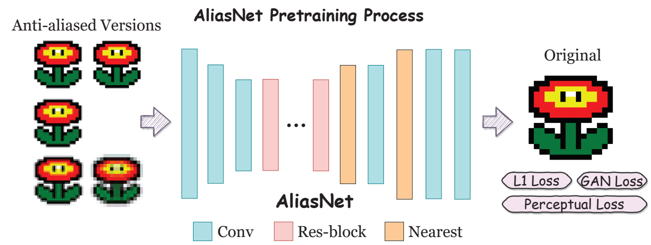


Fig. 11. The pre-training process of AliasNet. Using the original pixel arts and their anti-aliased versions in our Aliasing Dataset, the network learns to remove anti-aliasing effects and generate pixel arts with sharp edges. (© Nintendo Co., Ltd.)

output (see Fig. 11). The training process is similar to pix2pix [Isola et al. 2017]. In addition to the L1 loss and GAN loss, we also calculate the perceptual loss [Johnson et al. 2016] between the generated result and the target pixel art as part of the training loss function.

Pixel-Art-to-Image Network (P2INet). The P2INet converts a pixel art back to a non-pixel art image, to enable bi-directional and cyclical training of our network. It is a encoder-decoder network with the same structure as AliasNet (see Fig. 10 right). During testing, the P2INet can also be used for depixelizing pixel art images.

Numerical Encoder (NumEnc). The NumEnc module acts as a discriminator to ensure the result $\hat{p}_{c_i}$ generated by AliasNet has a pixel art appearance. To this end, we choose PatchGAN [Isola et al. 2017] as the architecture of NumEnc. During training, both the result $\hat{p}_{c_i}$ and the reference pixel art p_j are fed to NumEnc, and the outputs are used to calculate an adversarial loss (see Sec. 5.2). Although such an adversarial loss can help to generate results with pixel art appearance, the results may not have the desired cell size. To further enforce the cell size, we additionally use global average pooling and 1×1 convolution to convert the output of NumEnc into a cell feature vector that indicates the predicted cell size of the input image. The cell feature vector is then used to compute a *large margin cosine loss* that penalizes the deviation between the predicted cell size and the intended cell size (see Sec. 5.2 for details).

Discriminator for Non-Pixel Art Images. During the backward cycle of training, the image $\hat{c}_{p_j}$ generated by P2INet from a reference pixel art p_j needs to have a non-pixel art appearance. Similar to NumEnc, we introduce a PatchGAN discriminator to process both $\hat{c}_{p_j}$ and the input non-pixel art image c_i, and evaluate an adversarial loss (see Sec. 5.2).

5.2 Loss Function

In our network, the AliasNet and the structure extractor in CSEnc are pre-trained, while all other components are trained jointly. The latter training uses a loss function that consists of five components, as explained below.

Adversarial Loss. The adversarial loss forms a min-max game that guides I2PNet and P2INet to generate desired results. We utilize two PatchGAN discriminators [Isola et al. 2017] D_P and D_C, for the pixel art result $\hat{p}_{c_i}$ generated from c_i and the non-pixel art result $\hat{c}_{p_j}$ generated from p_j respectively (see Sec. 5.1), to discriminate whether they are true or fake via the following adversarial loss:

$$\begin{aligned}\mathcal{L}_D^{GAN} = &- \mathbb{E}_{c\sim\mathcal{C}}\left[\log D_C(c_i)\right] - \mathbb{E}_{p\sim\mathcal{P}}\left[\log D_P(p_j)\right] \\ &- \mathbb{E}_{c\sim\mathcal{C},p\sim\mathcal{P}}\left[\log(1 - D_C(\hat{c}_{p_j}))\right] \\ &- \mathbb{E}_{c\sim\mathcal{C},p\sim\mathcal{P}}\left[\log(1 - D_P(\hat{p}_{c_i}))\right].\end{aligned} \tag{2}$$

Accordingly, the two generators I2PNet and P2INet are trained via:

$$\mathcal{L}_G^{GAN} = -\mathbb{E}_{c\sim\mathcal{C},p\sim\mathcal{P}}\left[\log\left(D_C(\hat{c}_{p_j}) \times D_P(\hat{p}_{c_i})\right)\right]. \tag{3}$$

Cycle Consistency Loss. Zhu et al. [2017] first propose the cycle consistency loss to accomplish unsupervised image-to-image translation with unpaired data. We follow their approach and require the restored non-pixel art image c_i' in the forward cycle and the restored pixel art image p_j' in the backward cycle to be consistent with the input images c_i and p_j, respectively. This is achieved via the following loss:

$$\mathcal{L}_{cyc} = \frac{1}{|\mathcal{S}|}\sum\nolimits_{(c_i,p_j)\in\mathcal{S}}\left(\|c_i - c_i'\|_1 + \|p_j - p_j'\|_1\right), \tag{4}$$

where $\mathcal{S}$ is the set of training samples.

L1 Loss. To preserve the correct colors when translating a non-pixel art image c_i to a pixel art result $\hat{p}_{c_i}$, we apply the L1 loss to penalize their difference:

$$\mathcal{L}_{L1} = \frac{1}{|\mathcal{S}|}\sum\nolimits_{(c_i,p_j)\in\mathcal{S}}\|c_i - \hat{p}_{c_i}\|_1. \tag{5}$$

Identity Loss. It is challenging to reconstruct faithful colors in unsupervised image translation between different domains (e.g., non-pixel art to pixel art). Therefore, following [Taigman et al. 2017], we require our pixelization module (i.e., I2PNet followed by AliasNet) and depixelization module (i.e., P2INet) to be close to the identity map when applied to images from their respective target domains. This is enforced via the following identity loss:

$$\mathcal{L}_{idt} = \frac{1}{|\mathcal{S}|}\sum\nolimits_{(c_i,p_j)\in\mathcal{S}}\left(\|G_P(c_i) - c_i\|_1 + \|G_A(G_I(p_j)) - p_j\|_1\right), \tag{6}$$

where G_P, G_A, and G_I denote the P2INet, the AliasNet, and the I2PNet, respectively. The loss function prevents the networks from excessively changing the color of the output image.

Large Margin Cosine Loss. Our numerical encoder is used not only for adversarial training, but also for enforcing the same cell size between the generated pixel art $\hat{p}_{c_i}$ and the reference pixel art p_j. To do so, one possibility is to design a softmax cell size classifier based on the cell feature vector, and introduce a cross entropy loss for the generated and reference pixel arts. However, since the visual difference between different cell sizes may be inconspicuous, the cross entropy loss is not so effective in producing a clear decision boundary. In order to maximize the inter-class variance and minimize the intra-class variance, we instead utilize a Large Margin Cosine Loss (LMCL) that was originally used to distinguish similar faces [Wang et al. 2018]. The LMCL adds a cosine margin penalty to make the generated features more discriminative. Let f_{p_j} and $f_{\hat{p}_{c_i}}$ be the cell features produced by the numerical encoder for p_j and $\hat{p}_{c_i}$, respectively. Then we compute the LMCL based on the ground-truth cell size t_j of p_j as follows:

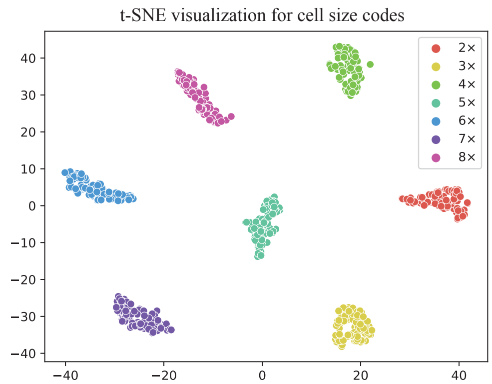


Fig. 12. After the training converges, the CSEnc module generates similar cell size codes for reference pixel arts in the same cell size. Here we apply CSEnc to 700 randomly chosen images from our Multi-cell Dataset, with 100 images for each cell size. The resulting cell size codes are visualized using t-SNE. We normalize each cell size code to a unit vector before applying t-SNE, since the weight modulation in Eq. (1) is scale-invariant with respect to the cell size code. The t-SNE visualization shows well-separated clusters of cell size codes for different cell sizes.

$$\mathcal{L}_{lmc} = \frac{1}{|\mathcal{S}|}\sum\nolimits_{(c_i,p_j)\in\mathcal{S}}\left(H_{t_j}(f_{p_j}) + H_{t_j}(f_{\hat{p}_{c_i}})\right). \tag{7}$$

Here the function $H_{t_j}(\cdot)$ is defined as

$$H_{t_j}(v) = -\log\left(\frac{e^{s\cdot(C_{t_j}(v)-m)}}{e^{s\cdot(C_{t_j}(v)-m)} + \sum_{k\neq t_j} e^{s\cdot C_k(v)}}\right), \tag{8}$$

where m is a margin parameter that is set to 0.4, s is a scaling factor that is set to 30, and

$$C_t(v) = \frac{W_t}{\|W_t\|}\cdot\frac{v}{\|v\|} \tag{9}$$

with W_t being a trainable weight vector associated with cell size t. Note that $C_t(\cdot)$ is the cosine distance from the weight vector for a particular cell size. The LMCL helps to enlarge the decision margin of cell size classification in the angular space for cell feature vectors, which provides more effective guidance for the network.

Overall Loss Function. In summary, the total loss for the entire network is a weighted sum of the above loss functions:

$$\begin{aligned}\mathcal{L}_{total} = &\lambda_{GAN}\left(\mathcal{L}_D^{GAN} + \mathcal{L}_G^{GAN}\right) + \lambda_{cyc}\mathcal{L}_{cyc} \\ &+ \lambda_{idt}\mathcal{L}_{idt} + \lambda_{L1}\mathcal{L}_{L1} + \lambda_{lmc}\mathcal{L}_{lmc},\end{aligned} \tag{10}$$

where λ_{GAN}, λ_{cyc}, λ_{idt}, λ_{lmc} are set to 1, 10, 10 and 1, respectively. λ_{L1} is set to 8 in the first 80 epochs, and increased to 10 afterward.

6 EXPERIMENTS

We conduct extensive experiments to demonstrate the effectiveness of our model and design choices, and compare its performance with existing pixelization methods.

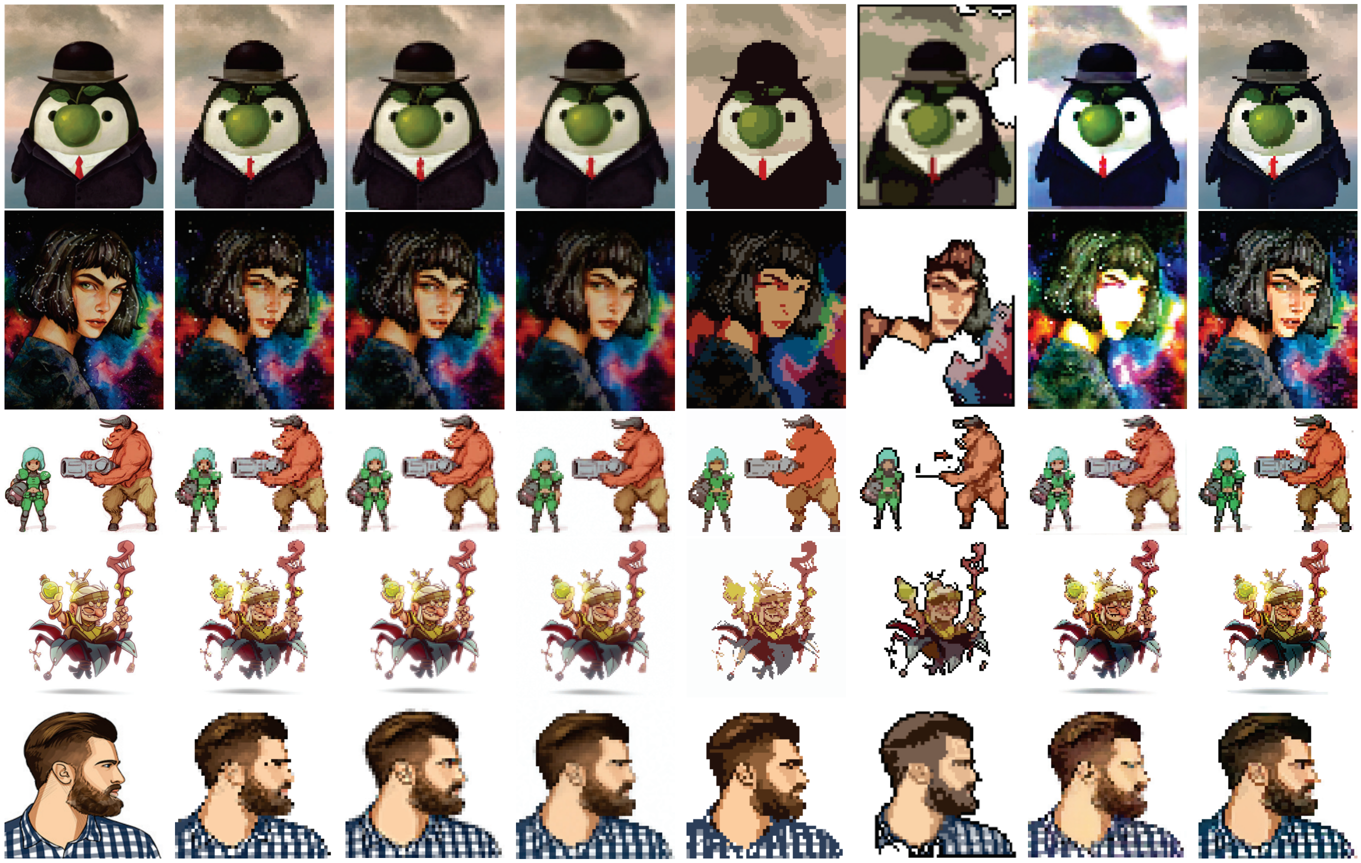


Fig. 13. Qualitative comparison of our method with state-of-the-art methods on pixelization task with the same cell size of 4×. Better viewed in digital version with zoom function. (© Tencent, © morganaOanagrom, © Arne, © Pablo Hernández and © Bee Square, © GOCARTOONME.)

Implementation Details. We implement our method using Pytorch [Paszke et al. 2019] on a single NVIDIA GeForce RTX 3090. To jointly train the network components, we use each image from the Multi-Cell Dataset as a reference, and pair it with a randomly chosen image from the non-pixel art dataset described in Section 4 as input. In total, we use 7000 pairs of input image and reference pixel art for the training. We train the structure extractor in CSEnc using SGD, while all other components in our network are trained using Adam [Kingma and Ba 2015] with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The training batch size is set to 2 and the learning rate is fixed to 0.0002 in all 160 epochs. Additionally, we adopt the training strategy of [Shrivastava et al. 2017], which uses historically generated images instead of the last one when the discriminators are updated. Therefore, we create two image buffers, each holding 50 generated images during training. Finally, we add spectral normalization [Miyato et al. 2018] to the numerical encoder to stabilize the GAN training process. It takes about four days for the training to converge.

During testing, we only need to use the I2PNet and AliasNet (for pixelization) or the P2INet (for depixelization). Note that although we need a reference pixel art to produce the cell size code during training, this is not necessary for testing: when the training converges, the CSEnc module produces similar cell size codes for different reference pixel arts with the same cell sizes, as shown in Fig. 12. Therefore, unless stated otherwise, we pre-select a representative cell size code for each cell size, and use them directly during testing. Also note that although we only use 512×512 input images for training, our model can handle input images of arbitrary resolution during testing. To test our method, we additionally collect 1000 non-pixel art images different from our training dataset. Besides the styles already included in the training set (e.g., cartoon clip arts), our testing set also covers other types of content such as portrait paintings, video game scenes and posters, and anime scenes. In addition, our method can convert a video into pixel art style by converting each frame. Thus we also collect several gameplay videos and test over 3000 frames. Unless stated otherwise, the results shown in this section are based on cell size 4×. Our method can efficiently convert an image into pixel art during testing: on average, it takes 0.052s to process a 192×192 image, and 0.344s for a 1280×720 video frame.

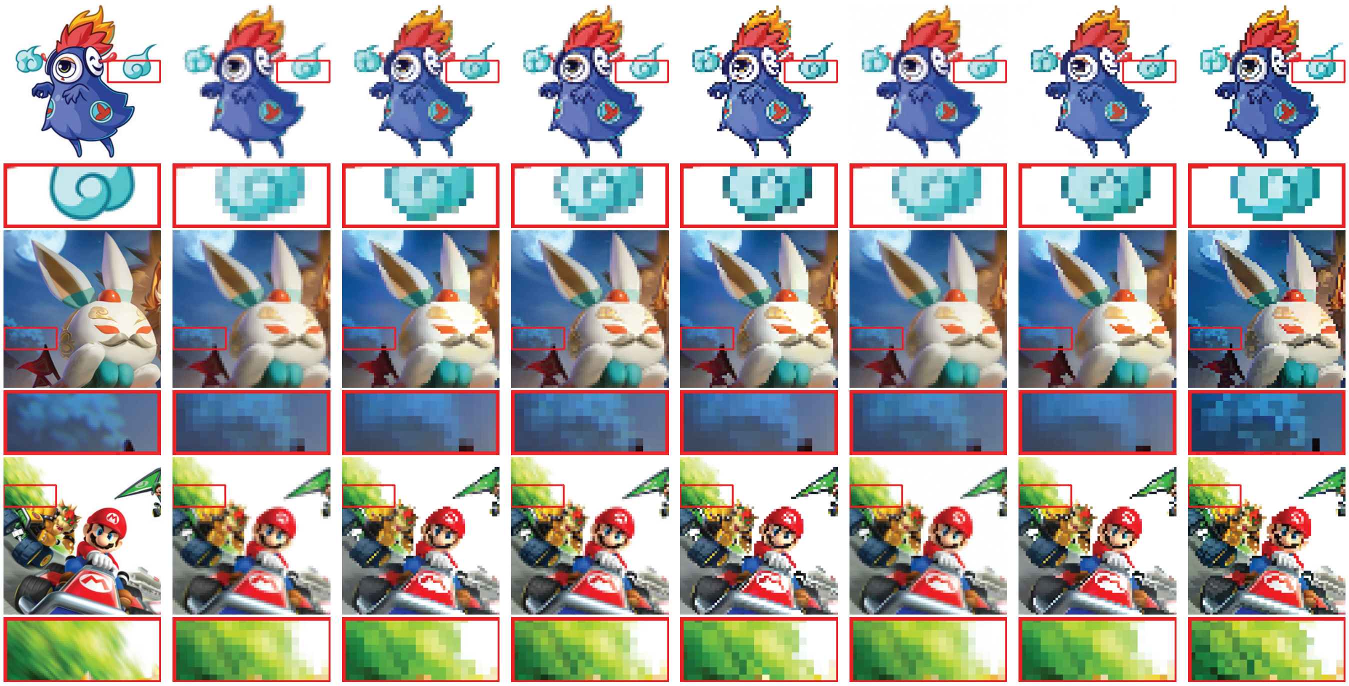


Fig. 14. Qualitative results obtained by using AliasNet to post-process results from different methods. (© Tencent, © Nintendo Co., Ltd.)

6.1 Qualitative Comparisons

For qualitative evaluation of the performance, we compare our model with several existing methods:

- non-learning-based methods, including nearest-neighbor downsampling (referred to as "*Nearest-neighbor*" in the following), [Öztireli and Gross 2015] ("*Perceptually-based*"), [Kopf et al. 2013] ("*Content-adaptive*"), and [Gerstner et al. 2012] ("*Image Abstraction*");
- learning-based methods, including [Han et al. 2018] ("*Deep Pixelization*"), and the online pixelization service "*PixelMe*" [Sato 2020] that is based on the pix2pix model [Isola et al. 2017].

For deep pixelization, the default model trained using the dataset in [Han et al. 2018] does not perform well in our experiments, since their dataset mainly consists of simple clip arts. Therefore, we retrain the model using our datasets to achieve better performance for a fair evaluation (see the supplementary materials for comparisons between the original model and our retrained one). We use each method to generate a pixel art with the same cell size and in the same resolution as the input image. For methods that generate a downscaled result smaller than the input, we choose an appropriate downscaling ratio to generate a result and then upscale it to the same resolution as the input using nearest-neighbor interpolation.

Fig. 13 shows a comparison between different methods for cell size 4×. As a basic downsampling method, *Nearest-neighbor* can lose important details and result in blocky and discontinuous edges. As a non-linear filtering approach, *Perceptually-based* produces blurry results with anti-aliasing effects on the edges. Although *Content-adaptive* generates results with sharper edges than *Perceptually-based*, it still suffers from anti-aliasing effects and can lose crucial details as a kernel-based approach. *Image Abstraction* is able to generate crisper results and mitigate the anti-aliasing effects, but it limits the scale of the color palette after quantization, leading to discontinuity on edges that is inconsistent with the input (e.g. see the third row). As for *PixelMe*, it tends to wrap the pixelization result with a continuous black boundary, so that the image has continuous edges. However, *PixelMe* detects the image content automatically before conducting pixelization, which can lead to loss of features if the detection is inaccurate. For example, in the first two rows, some background regions are missing in the final results. *Deep Pixelization* shows quite good results with crisp edges and less ringing effect. However, some results have notable color distortion, due to their winner-take-all loss function, the mandatory downsampling operations in their model design, and the entangled cell structure and colors in their training. In comparison, our results are much better with crisp and continuous edges, avoiding ringing and blurring effects. Our model also preserves local details of original inputs.

Sharp edges and aliasing appearances are important characteristics of pixel art. Our AliasNet not only helps to produce such desirable appearance in our network, but also can be used as a standalone post-processing tool to improve pixelization results that suffer from anti-aliasing effects. In Fig. 14, we apply the AliasNet to some results produced by bicubic downsampling, *Perceptually-based*, and *Content-adaptive*. All three methods suffer from anti-aliasing

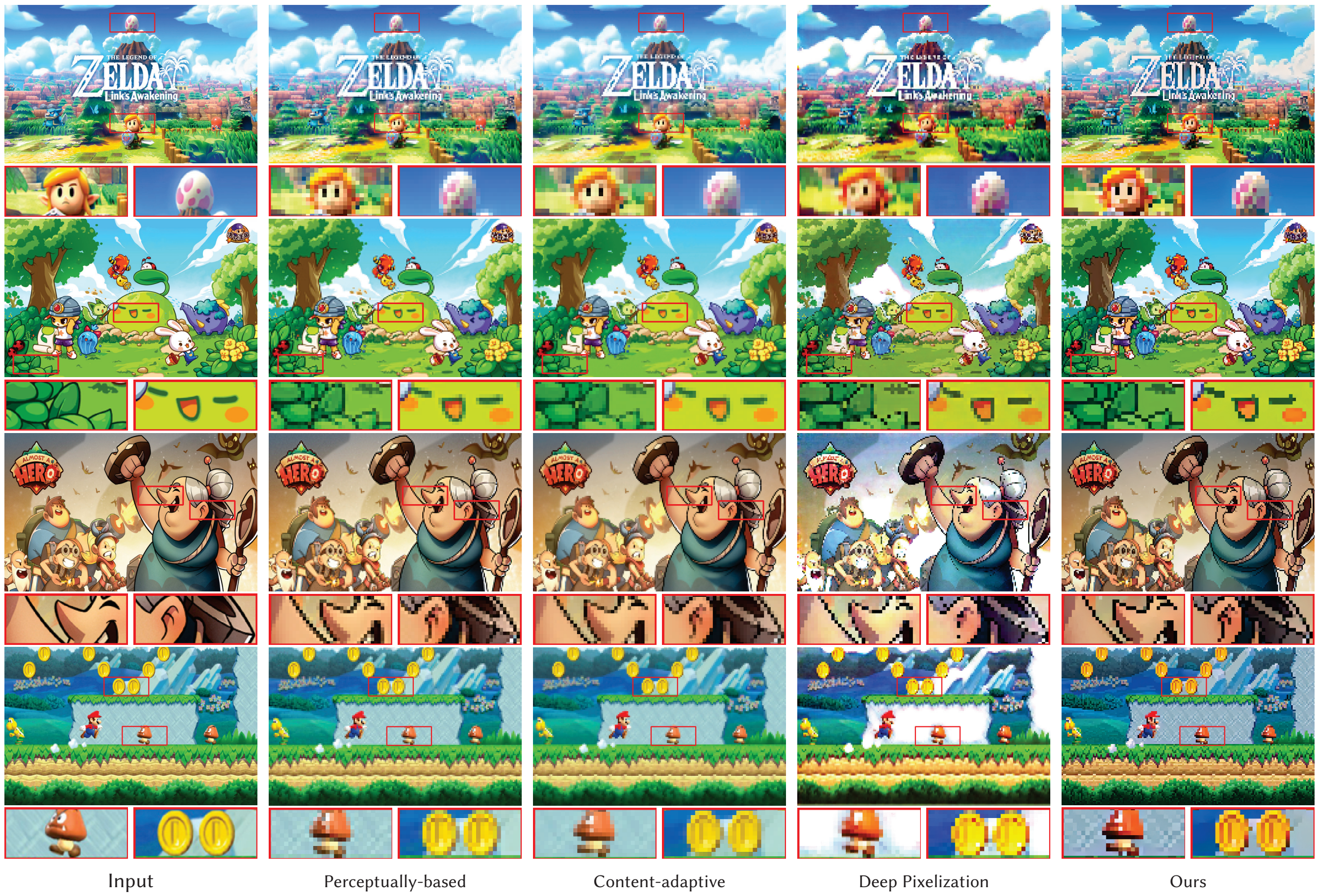


Fig. 15. Qualitative comparison on game scenes. Better viewed in digital version with zoom function. (© Nintendo Co., Ltd., © Tencent, © Bee Square.)

effects in their results, and AliasNet helps to improve the sharpness of the edges. On the other hand, some details from the input have already been removed by these methods and cannot be recovered even with AliasNet (e.g., see the zoomed region in the second and third rows). In comparison, our method produces sharp pixelated appearance while preserving the details.

Since pixel art is often used in games, we further compare our model with existing work on game scenes, and show the results in Fig. 15. The testing images are in a resolution of 680×425 or 824×516 and obtained from game screenshots and posters. Because of the limited input resolution required by *Image Abstraction* and *PixelMe*, we omit these two methods in this experiment. From Fig. 15, we can see that the performance of different methods is rather similar to Fig. 13, and our model can generate good results with the desirable pixelated appearance even for higher resolution images. This also demonstrates the possibility of using our method to convert high-quality video games into pixel art style, by simply converting each frame individually. Some qualitative results of such video pixelization can be found in the supplementary video.

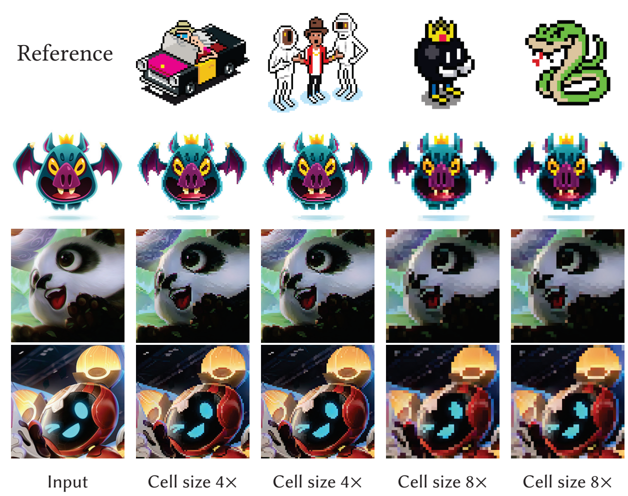


Fig. 16. Results from our model using reference pixel arts in different cell sizes. Our method can transfer the cell size style of the references into pixelization result, regardless of the content in the references. (© eBoyArts, © Pablo Hernández and © Bee Square, © Tencent.)

Fig. 17. Qualitative comparison between different methods on high-resolution input images with different cell sizes of pixelization results. Note that the cell size settings of *Deep Pixelization* are different from other methods, because the cell size choices are hard-coded in the model design and cannot be changed. Better viewed in digital version with zoom function. (© Tencent.)

6.2 Cell-controllable Pixelization

One benefit of our model is that it can generate pixel arts with a specific cell size. To demonstrate the controllability of cell size, in Fig. 16 we pixelize three input images using cell size codes generated by CSEnc from four reference pixel arts, where two references have cell size 4× and the other two have cell size 8×. We can see that our method generates results with a cell size consistent with the reference pixel art. In addition, reference pixel arts with the same cell size lead to almost identical pixelization results. This is because for pixel arts with the same cell size, the CSEnc module generates similar cell size codes regardless of their content (also see Fig. 12). The results show that our method can effectively transfer the cell size style from the reference pixel art to the resulting image.

In Fig. 17, we compare our method with *Perceptually-based*, *Content-adaptive* and *Deep Pixelization* in terms of cell controllability, using input images at the resolution of 960×600 and 480×720, respectively. For *Perceptually-based* and *Content-adaptive*, with a small cell size of 2× the pixelization effects are less visible than the other two methods, while at larger cell sizes of 4× and 6× their results suffer from anti-aliasing effects with a blurry appearance. As for *Deep Pixelization*, it uses a winner-take-all strategy in the loss function and is only well trained for smaller cells, since a smaller cell size is closer to the ground truth and tends to lower the loss function. Moreover, producing a pixel art appearance while assigning the correct flat colors to the cells is a highly ambiguous training process. *Deep Pixelization* learns these two functions together, leading to unstable color assignments as seen in the results. In comparison, the separated learning of cell size features and color assignment in our model helps to overcome these limitations, simultaneously maintaining regular cell structures, significant aliasing effects and accurate color assignments during the pixelization process.

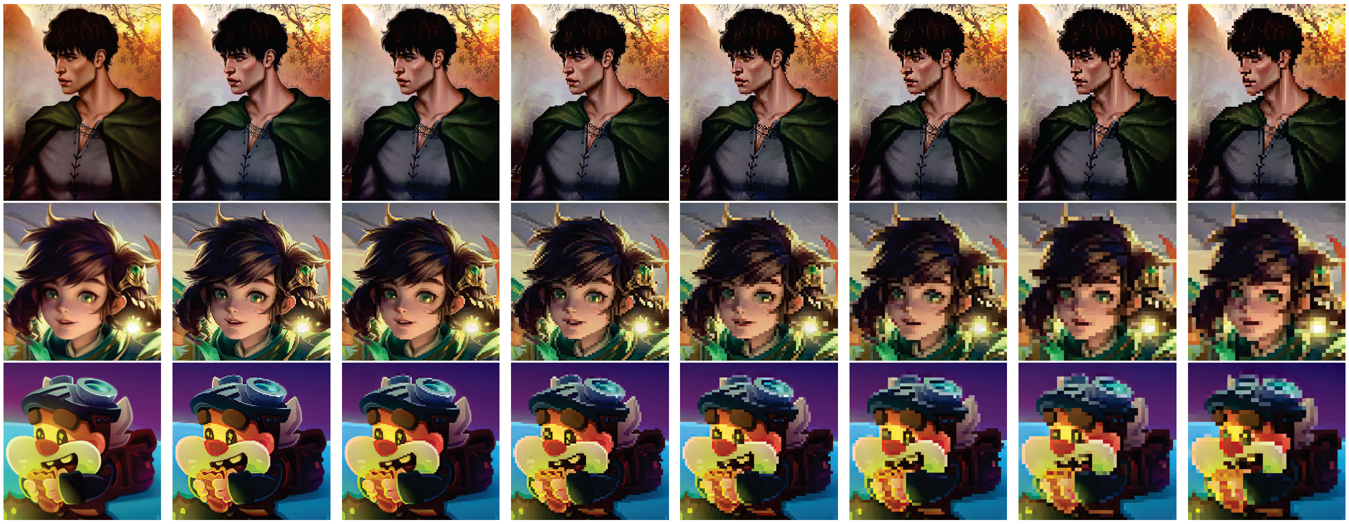
Table 1. Quantitative evaluation under two metrics (i.e. FID and KID). The best results are highlighted in **bold**.

Method	$FID\downarrow$	$KID\times10^2\downarrow$
Perceptually-based [Öztireli and Gross 2015]	217.04	9.20
Content-adaptive [Kopf et al. 2013]	181.66	5.99
Image Abstraction [Gerstner et al. 2012]	169.46	5.24
PixelMe [Sato 2020]	189.54	6.64
Deep Pixelization [Han et al. 2018]	181.63	5.73
Ours	**165.87**	**4.24**

Fig. 18 further demonstrates results from our model for cell sizes from 2× to 8×. More results can be found in the supplementary materials. The results show that our method is effective across different input image styles and different cell sizes.

6.3 Quantitative Comparison

We also compare different methods using two metrics: Fréchet Inception Distance (FID) [Heusel et al. 2017] and Kernel Inception Distance (KID) [Bińkowski et al. 2018], which are commonly used in image generation tasks [Karras et al. 2019, 2020; Park et al. 2019]. FID measures the Fréchet distance between two Gaussian distributions obtained from the generated images and the real ones respectively, where a lower FID indicates a higher similarity. KID is similar to FID, but is computed by the squared Maximum Mean Discrepancy (MMD) between Inception features extracted from the generated and real images. Since KID has a simple unbiased estimator, it is more consistent with human perception. Tab. 1 shows the values of FID and KID using different methods on the testing dataset. Our method outperforms the other methods by a large margin, indicating that we can generate more realistic pixelization arts.

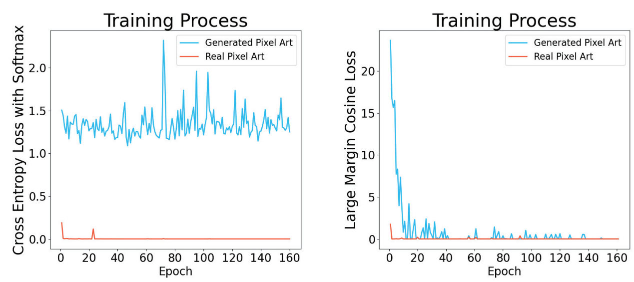


Input 2× 3× 4× 5× 6× 7× 8×

Fig. 18. More results from our method in different cell sizes. Better viewed in digital version with zooming. (© morganaOanagrom, © Tencent, © Pablo Hernández and © Bee Square.)

Input I2PNet output AliasNet output P2INet output

Fig. 19. Stage-wise results of our network. (© Pablo Hernández and © Bee Square, © Rendered Ideas.)



Softmax LMCL

Fig. 20. Comparison between training loss using LMCL (eq.(7)) and softmax-based cross entropy loss, respectively. *Softmax* encounters convergence problems for the generated pixel arts, while LMCL converges successfully and produces pixel art results that can fool the discriminators.

6.4 Ablation Study

We further conduct ablation studies to evaluate the effectiveness of the components in our pipeline.

Effectiveness of Two-stage Pixelization. The two-stage pixelization process (i.e., the use of I2PNet followed by AliasNet) is a key factor for the success of our model. To investigate what is learned in different sub-modules, we visualize the intermediate results generated by our network in Fig. 19. The input image is passed through I2PNet to obtain an intermediate result image that shows a cell structure but with degraded color appearance. The artifacts near the edges can be observed when zooming into the images at this stage. After passing through AliasNet, the image is refined into visually pleasing pixelization results. The pixelization results are then reconstructed back to the non-pixel art form by P2INet to enable cyclical training.

The underlying cause for this two-stage effect is that AliasNet tends to reverse the anti-aliasing in an exaggerated way because of the synthetic training data (i.e., the Aliasing Dataset). As anti-aliasing itself can soften the edges and diminish the color tone, the synthetic input data also suffers from this problem by mimicking this reverse process. Thus, AliasNet is trained to restore the degraded color. Accordingly, our I2PNet learns to darken the result, such that AliasNet can produce high-quality results.

Effectiveness of LMCL. To examine the effectiveness of the LMCL component (7) in our loss function, we compare it with a more general baseline loss, which is the cross entropy loss (*Softmax*). We visualize the training loss of real pixel arts and generated pixel arts for these two losses separately and show them in Fig. 20. It can be seen that *Softmax* converges quickly on real pixel arts, but has difficulties converging on generated results. Instead, LMCL mitigates

Fig. 21. Ablation study on various model design choices. (© Pablo Hernández and © Bee Square, © Tencent, © Nintendo Co., Ltd.)

this problem and obtain rather low loss values as training goes for both types of pixel arts. It shows that with LMCL, the generated pixel arts can fool the discriminators successfully, and their distribution is close to the real ones. The results also imply that *Softmax* has trouble producing pixelization results in our task.

Analysis of Architecture Variants. We set up several variants of our network to evaluate the key components in its architecture:

- *CycleGAN*: this is a single stage process where the input is sent into I2PNet for generating a pixelization result and then converted back through P2INet;
- *Encoder only*: it replaces the decoder part of I2PNet with nearest-neighbor upscaling operator based on *CycleGAN*;
- *Baseline*: it further adds the AliasNet based on *CycleGAN*;
- *Concat*: rather than encoding the reference through CSEnc, the reference image is concatenated with the input as the input to I2PNet;
- *Without $\mathcal{L}_{L1}$*: it removes the $\mathcal{L}_{L1}$ from the loss function of our full model;
- *Without $\mathcal{L}_{lmc}$*: it removes the $\mathcal{L}_{lmc}$ from the loss function of our full model;
- *Softmax*: it replaces $\mathcal{L}_{lmc}$ with the softmax-based cross entropy loss for the loss function of our full model;
- *One-hot*: rather than using CSEnc to derive the cell size code from a reference pixel art, we feed a one-hot vector for the cell size into an MLP to produce the cell size code.

We show a qualitative comparison between these variants in Fig. 21. Purely relying on *CycleGAN* generates mild pixelization effects but introduces white speckle artifacts (e.g., see the region near the ear in the top row). Replacing the decoder part of I2PNet with nearest-neighbor operator (*Encoder only*) is not effective in producing pixelization effects but results in artifacts and over-saturated colors, which demonstrates the importance of our encoder-decoder structure as well as the cell size code embedding. After adding the AliasNet (*Baseline*), the model gets rid of speckle artifacts, the pixelization effects are still not obvious. Simply adding the reference through concatenation (*Concat*), the result may not show a pixelization effect in all regions (e.g., see the legs in the second row), which indicates the benefit of our CSEnc in directly encoding the reference image and extracting structural features. For variants related to the loss functions, removing $\mathcal{L}_{L1}$ may result in color distortions (e.g., see the background color in the second row). Removing $\mathcal{L}_{lmc}$ can pixelize the outermost edges of the input image but fails in the details (e.g., see the mouth in the top row). The result of *Softmax* further validates our finding in Fig. 20. Using a one-hot vector for the cell size rather than a reference image, the model can still generate pixelated appearance but with serious artifacts (e.g., see the background in the second and third rows). Different from these variants, our full model is able to generate visually pleasing and crisp pixel arts that align with the inputs.

6.5 Depixelization

Since our model adopts a cycle training strategy, it can not only pixelize natural images but also depixelize pixel arts. More specifically, the depixelization process is conducted with P2INet, as it is trained to restore pixel arts back to smooth and high-quality images. Fig. 22 shows some depixelization results using P2INet on some pixel arts outside the training set, and compares it with the state-of-the-art approach from [Kopf and Lischinski 2011]. Our method is able to generate good-quality results, with smooth and clear details comparable to [Kopf and Lischinski 2011].

6.6 Limitations

Although our model can achieve good results, it still has some limitations. First, our model does not enforce a strict requirement for the $N \times N$ pixels inside a cell to have the exact same color; as a result, there might be a slight difference between the pixel colors within a cell. In our experiments, such color difference is not noticeable in general; it can also be easily fixed by using nearest-neighbor downsampling to reduce the result into a one-cell-one-pixel form and then upscaling it to the correct cell size (see Fig. 24 for some examples). Second, to ensure color fidelity during pixelization, our model is trained to preserve the features from the input image as much as possible. A side effect is that noises from the input may be treated as features and retained in the output. As shown in Fig. 23, some existing works such as *Perceptually-based* and *Content-adaptive* can

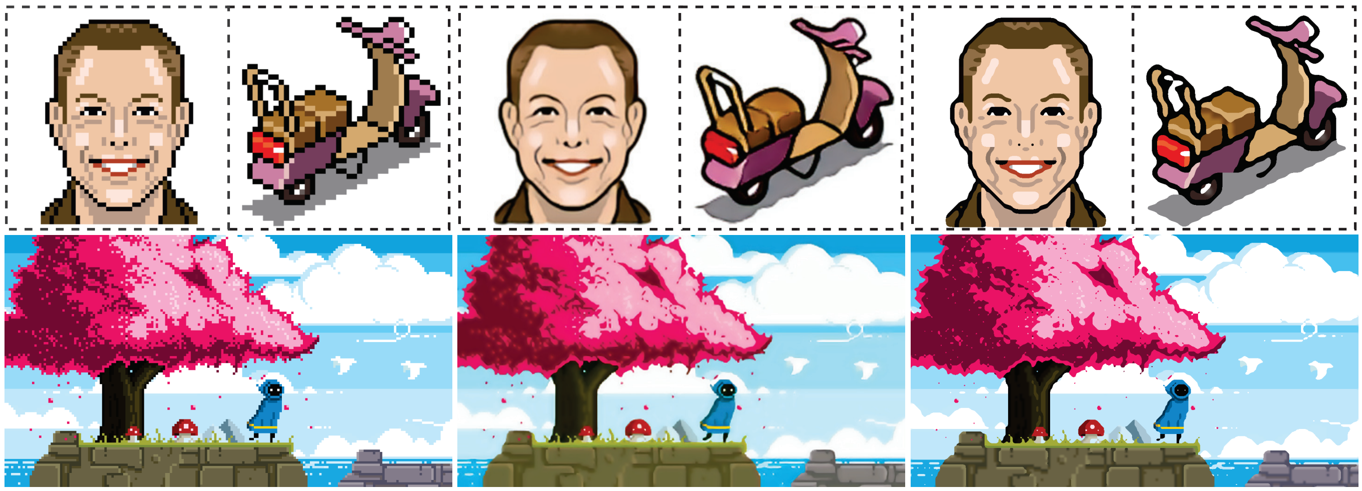


Fig. 22. Depixelization results enabled by our framework. Given the cyclical nature of our model, we can use P2INet to perform depixelization on a pixel art to generate a natural image. (© eBoyArts, © PXLFLX/DeviantArt.)

Fig. 23. Unlike *Perceptually-based* and *Content-adaptive*, our method may treat the noises in the input as features and retain them in the output pixel art. (© Tencent.)

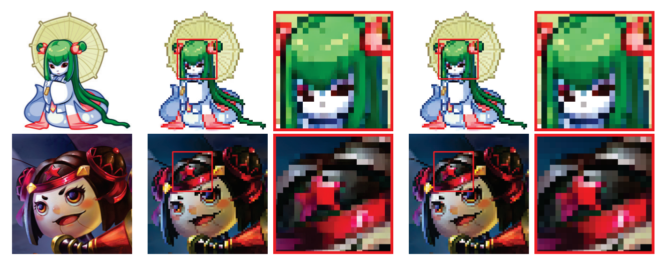


Fig. 24. Our model does not strictly enforce the same color within each cell. If needed, this can be easily enforced by post-processing using nearest-neighbor downsampling and upscaling. In general, the images generated by our model already achieves nearly constant color in each cell, and the post-processing makes little visual difference. (© Tencent.)

filter out most of the noises due to the nature of their formulation, while our model preserves these noises and generates unsatisfactory results. One possible solution is to introduce noisy inputs into the training data. Third, our model utilizes the L1 loss to enforce color fidelity between input and final output. This works well in most cases, but a slight color shift may still arise in a small number of results (e.g., the last row of Fig. 15). The results may be improved by designing a more sophisticated color fidelity loss, which will be left as future work.

7 CONCLUSION

This work presents a novel data-driven approach for generating high-quality pixel arts from images in a cell-controllable manner. To enable such control in cell size, our model takes a reference pixel art as an additional input during training, to provide structural and numerical regularization. More specifically, the reference pixel art is encoded into a feature vector that indicates the cell size, which is fused into a two-stage pixelization process to guide the cell style of the resulting image. The two-stage pixelization is disentangled to cell-aware and aliasing-aware stages, resolving the ambiguities between cell structure, aliasing effects, and color assignment in traditional joint training approaches. A large margin cosine loss is introduced into the loss function, to penalize the cell size difference between the generated pixel art and the reference. A dataset of "one-pixel-one-cell" pixel arts, together with its multi-cell and anti-aliasing augmentations, are utilized to train our model. Through extensive quantitative and qualitative experiments, we have shown the effectiveness of our model on diverse types of images for generating high-quality pixelization results with sharp edges and faithful expressive contents. Besides, we have demonstrated that our model can deal with high-resolution images, and we provide the first feasible solution for video pixelization.

ACKNOWLEDGMENTS

This project is supported by the National Natural Science Foundation of China (No. 61972162); Guangdong International Science and Technology Cooperation Project (No. 2021A0505030009); Guangdong Natural Science Foundation (No. 2021A1515012625); Guangzhou

Basic and Applied Research Project (No. 202102021074); and CCF-Tencent Open Research fund (CCF-Tencent RAGR20210114).

REFERENCES

Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. 2018. Demystifying MMD GANs. In *ICLR*.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*. 248–255.

Ruigang Fu, Qingyong Hu, Xiaohu Dong, Yulan Guo, Yinghui Gao, and Biao Li. 2020. Axiom-based grad-cam: Towards accurate visualization and explanation of cnns. In *BMVC*.

Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam Gingold, and Andrew Nealen. 2013. Pixelated image abstraction with integrated user constraints. *Computers & Graphics* 37, 5 (2013), 333–347.

Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam I Gingold, and Andrew Nealen. 2012. Pixelated image abstraction. In *NPAR@ Expressive*. 29–36.

Chu Han, Qiang Wen, Shengfeng He, Qianshu Zhu, Yinjie Tan, Guoqiang Han, and Tien-Tsin Wong. 2018. Deep unsupervised pixelization. *ACM TOG* 37, 6 (2018), 1–11.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, Vol. 30.

Xun Huang and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*. 1501–1510.

Tiffany Inglis and Craig S Kaplan. 2012. Pixelating vector line art. In *SIGGRAPH Posters*. 108.

Tiffany Inglis, Daniel Vogel, and Craig S Kaplan. 2013. Rasterizing and antialiasing vector line art in the pixel art style. In *NPAR*. 25–32.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *CVPR*. 1125–1134.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*. Springer, 694–711.

Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4401–4410.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and improving the image quality of stylegan. In *CVPR*. 8110–8119.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Johannes Kopf and Dani Lischinski. 2011. Depixelizing pixel art. In *ACM SIGGRAPH 2011 papers*. 1–8.

Johannes Kopf, Ariel Shamir, and Pieter Peers. 2013. Content-adaptive image downscaling. *ACM TOG* 32, 6 (2013), 1–8.

Hailan Kuang, Nan Huang, Shuchang Xu, and Shunpeng Du. 2021. A Pixel image generation algorithm based on CycleGAN. In *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Vol. 4. IEEE, 476–480.

Ming-Hsun Kuo, Yong-Liang Yang, and Hung-Kuo Chu. 2016. Feature-Aware Pixel Art Animation. In *Computer Graphics Forum*, Vol. 35. 411–420.

Junjie Liu, Shengfeng He, and Rynson W. H. Lau. 2018. L_0 -Regularized Image Downscaling. *IEEE TIP* 27, 3 (2018), 1076–1085.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957* (2018).

A. Cengiz Öztireli and Markus Gross. 2015. Perceptually based downscaling of images. *ACM TOG* 34, 4 (2015), 1–10.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. 2019. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2337–2346.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, Vol. 32. 8026–8037.

Amélie Royer, Konstantinos Bousmalis, Stephan Gouws, Fred Bertsch, Inbar Mosseri, Forrester Cole, and Kevin Murphy. 2020. Xgan: Unsupervised image-to-image translation for many-to-many mappings. In *Domain Adaptation for Visual Understanding*. Springer, 33–49.

Sato. 2020. PixelMe: Convert your photo into pixelart. https://pixel-me.tokyo/en/.

Yunyi Shang and Hon-Cheng Wong. 2021. Automatic portrait image pixelization. *Computers & Graphics* 95 (2021), 47–59.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. 2017. Learning from simulated and unsupervised images through adversarial training. In *CVPR*. 2107–2116.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Yaniv Taigman, Adam Polyak, and Lior Wolf. 2017. Unsupervised cross-domain image generation. In *ICLR*.

Ken Turkowski. 1990. Filters for common resampling tasks. *Graphics Gems I* (1990), 147–165.

Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. 2018. Cosface: Large margin cosine loss for deep face recognition. In *CVPR*. 5265–5274.

Nicolas Weber, Michael Waechter, Sandra C Amend, Stefan Guthe, and Michael Goesele. 2016. Rapid, detail-preserving image downscaling. *ACM TOG* 35, 6 (2016), 1–6.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*. 2223–2232.

C Lawrence Zitnick and Devi Parikh. 2013. Bringing semantics into focus using visual abstraction. In *CVPR*. 3009–3016.